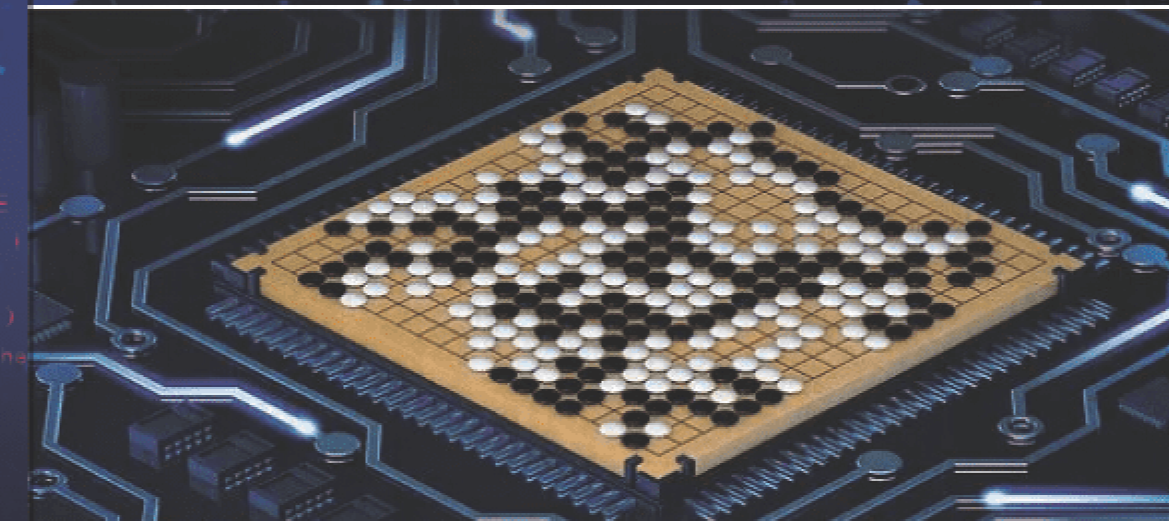
AlphaGo如何能战胜顶级围棋选手，本书将给你揭开神秘的面纱



AlphaGo如何战胜人类围棋大师

智能硬件TensorFlow实践

- 本书理论与实践结合，带领你动手学习和了解最热门的**人工智能技术**
- 通过深度学习TensorFlow工具的实践，处理真实世界的人机交互问题
- 本书带你快速入门人工智能技术，通过实例说明让硬件如何**“智能化”**

陈震　郑文勋◎编著

清华大学出版社

AlphaGo如何战胜人类围棋大师

——智能硬件TensorFlow实践

陈　震　郑文勋　编著

清华大学出版社
北　京

内 容 简 介

本书主要阐述了当前机器智能的热点技术——深度学习和强化学习技术的原理。在此基础上，介绍 AlphaGo 结合深度学习和强化学习技术，如何战胜人类围棋大师的原理。接下来，将深度学习的理论转化为实践，给出如何通过掌握 TensorFlow 和 Keras 深度学习框架，制作声控智能硬件的例子，同时给出机器视觉的对象检测案例，指导读者逐步学习使用深度学习技术。

本书的主要特点是实践操作，用实用可运行的案例来上手。本书可作为实践入门指导书，适用于对机器智能有兴趣的高年级本科生，也适合于对机器智能有兴趣的人员参考。

图书在版编目(CIP)数据

AlphaGo 如何战胜人类围棋大师：智能硬件 TensorFlow 实践/陈震等编著. —北京：清华大学出版社，2018
ISBN 978-7-302-49270-2

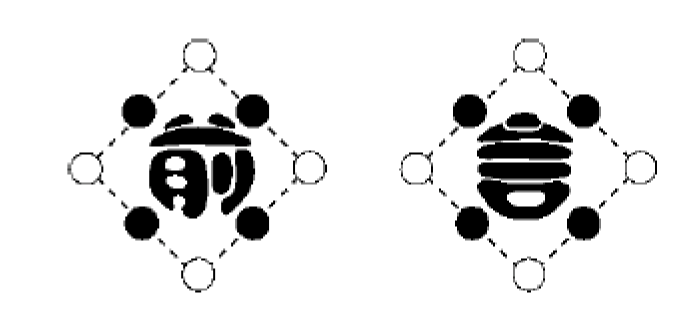
Ⅰ. ①A…　Ⅱ. ①陈…　Ⅲ. ①人工智能—算法—研究　Ⅳ. ①TP18

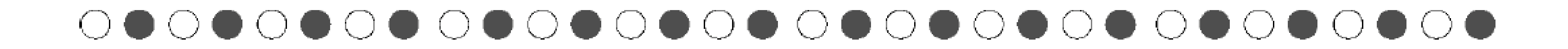
中国版本图书馆 CIP 数据核字(2018)第 003476 号

责任编辑：白立军
封面设计：杨玉兰
责任校对：梁　毅
责任印制：沈　露

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 装 者：北京鑫海金澳胶印有限公司
经　　销：全国新华书店
开　　本：185mm×230mm　　**印　　张**：8.25　　**字　　数**：147 千字
版　　次：2018 年 7 月第 1 版　　**印　　次**：2018 年 7 月第1 次印刷
印　　数：1～2000
定　　价：39.00 元

产品编号：075982-01





信息技术日新月异，机器智能更是一个快速发展的领域，其所引发的社会变化和带来的社会影响也是巨大的。围绕这一领域的热点技术，如深度学习和强化学习，涉及一些基础数学知识，包括微积分、线性代数和优化理论等。机器智能作为计算机科学的一个应用，虽然涉及计算机体系结构、分布式系统、软硬件协同设计、算法与数据管理等诸多计算机理论知识，但其核心内容还是算法与数据。

AlphaGo 战胜人类围棋大师，其实就是人类所创造的智能工具能力的胜利，是科学理性的胜利。AlphaGo 的成功，证明了基于深度强化学习和蒙特卡洛树搜索方法的机器智能，在很多规则清晰的场景下完全可以比人类做得更好。

机器智能的领域不断扩大，遇到的问题也越来越多，需要人们不断创新，不断深入探究。近些年来的教学实践表明，对于一个新的知识领域，在具备一定的基础知识后，本科生完全有能力投入这一领域前沿技术的研究工作中。本书的目的就是要通过系统整理机器智能领域知识点，帮助本科同学迅速了解全貌，从而快速深入技术细节，为进一步的科研工作打下基础。

大凡与机器智能相关的技术，都需要训练有素的头脑，快速分析问题与解决问题的能力。所以，本科同学要想进入这个领域，除了解、掌握本书中的知识和实践操作外，还需要不断地训练自己思考问题和解决问题的能力。

本书的编写离不开清华大学 iCenter 智能系统实验室教师团队的协助，他们是马晓东、章屹松、王蓓和高英。本书在本科生课程“大数据智能”与“智

能硬件"的实践教学中，根据反馈意见已经做了多次修订。实验室学生郑文勋、王亦凡、常嘉辉、吴垠鋆、冯杰、宋丹丹、钱鹏等多次担任课程或单元的助教，为本书的完善做出很大贡献。实验室SRT学生的多次学术活动，也为本书提供了有益的参考资料。

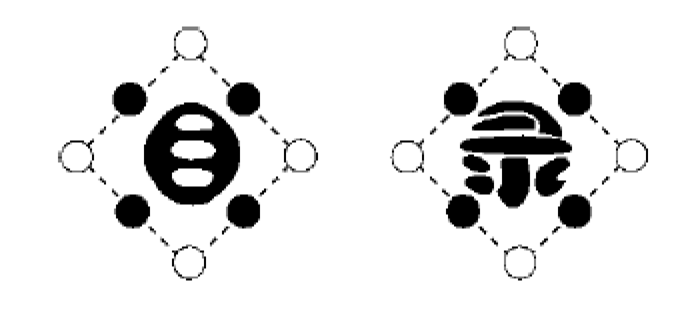
同时，我们也得到微软公司ETG团队的大力支持，他们是杨滔、章艳、刘士君、闫伟。微软公司除了提供云计算与机器学习服务支持外，还连续三届为清华iCenter——人工智能挑战赛提供了支持，极大方便了我们的课程教学和实验工作。

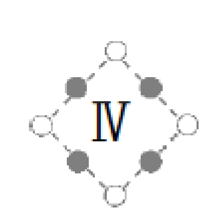
最后，感谢所有参与我们课程及挑战单元的同学们。他们朝气蓬勃，锐意进取，对未知领域充满好奇并进行着不知疲倦的探索。我们坚信，他们是学术和产业的希望及未来。

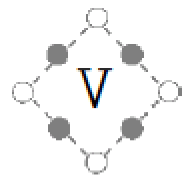
书中代码可从清华大学出版社网站www.tup.com.cn下载。

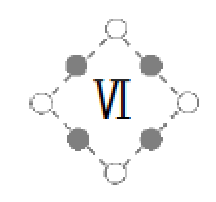
作　者

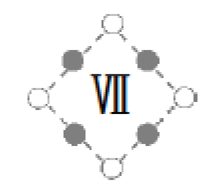
2018年1月


目 录




V


Ⅵ







1.1 机器智能

1.1.1 机器智能的定义

通俗地说，机器智能就是用机器（如计算机）完成人类需要用大脑完成的任务，代替人脑的工作，例如下棋、开车、阅读理解等。

机器智能是指计算机系统体现的智能的能力，如从听、说、读、写到搜索、推理、决策和回答问题等，同时也指如何设计实现计算机系统和软件，使其具有智能的行为。

1.1.2 机器智能的分类

目前，机器智能一般分为机器感知和机器认知两个层面，如图 1-1所示。

机器感知：包括语音识别、视觉识别、运动控制和眼手协同等。

机器认知：包括机器学习、自动推理、人工意识和知识表示等。

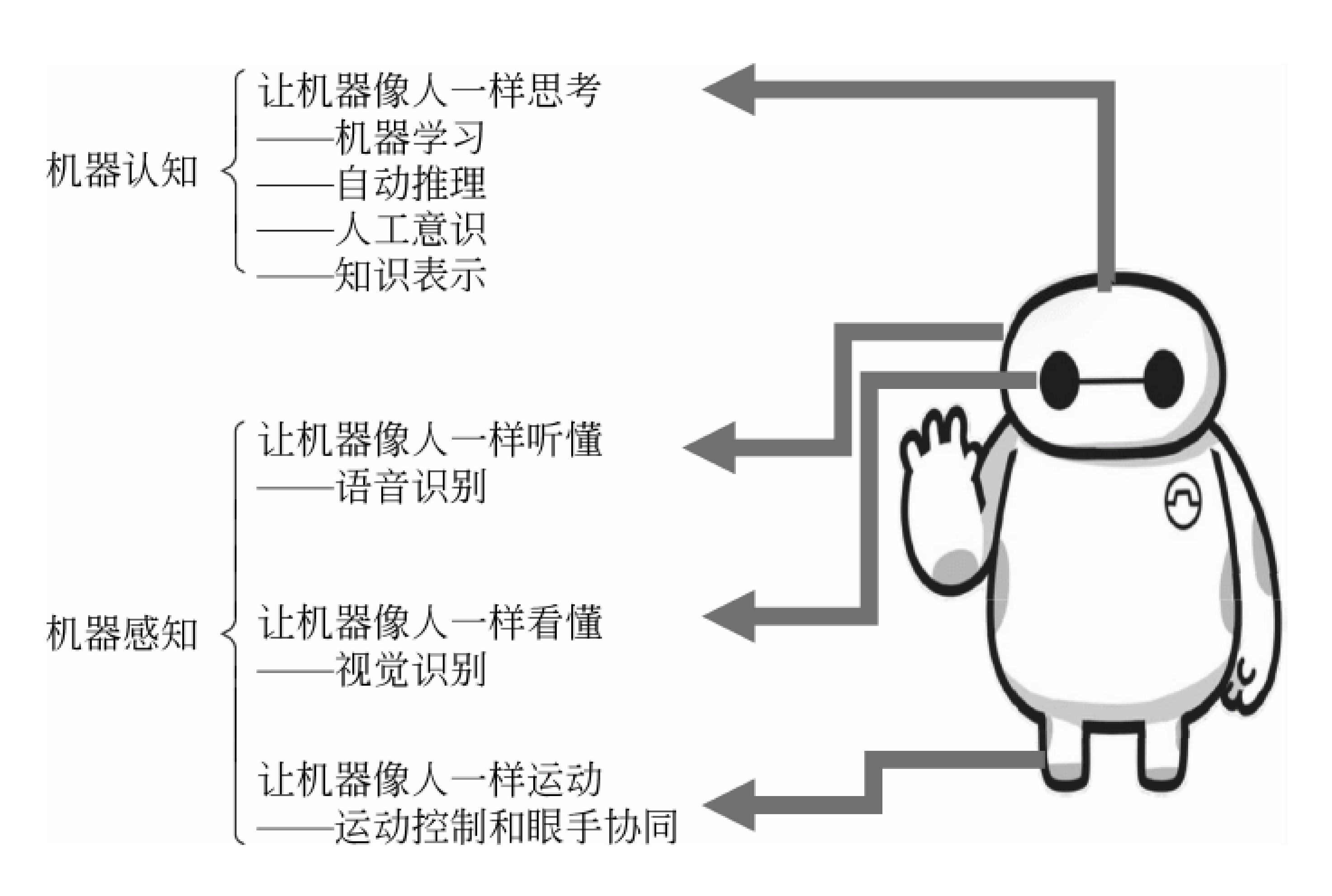


图 1-1　机器智能示意图

1.2 深度学习

1.2.1　机器智能的神经网络方法

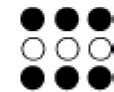
人脑含有约 860 亿个神经元，还有大致 850 亿个非神经细胞。在大脑皮层(Cortex)约有 160 亿个神经元，小脑有 690 亿个神经元。人脑的神经元一般只与周围几千个神经元相连接，彼此能够传导电信号。

要让机器代替人脑工作，最直观的方法就是用机器来模拟人脑的工作行为。但是要模拟这些人脑工作的原理有难度。

首先，人脑作为一个系统，太复杂，涉及生物学、生理学、化学等学科知识；其次，至今人们还未彻底理解人脑的各个功能的工作机理。当然，确实有研究人员是沿着这个思路去实现机器智能的。

那是否还有别的思路呢？其实思路也挺多的，这里有一种称为人工神经网络(Artificial Neural Networks)的方法，最近在实践中被证明是有效的方法。这种方法又称为连接主义(Connectionism)，其核心思想是通过大量简单计算单元连接起来的网络来实现复杂的智能行为。

这种方法首先是用数学方法来抽象出单个神经元(Neuron)的功能，构建出单个人工



神经元的模型。其次，在单个神经元建模的基础上，参考人脑中神经元的连接方式，构建人工神经网络。最后，通过输入数据样本给神经网络训练，调整神经网络的参数，使其完成具有某些智能的任务（如眼睛看、耳朵听等）。

1.2.2 人工神经元与人工神经网络

对于单个神经元的活动原理，目前已经有比较深入的研究。不论何种神经元，从功能上可以分为接收区域（Receptive Zone）、触发区域（Trigger Zone）、传导区域（Conducting Zone）和输出区域（Output Zone）。

这里给出人工神经元的数学抽象模型，也称为逻辑斯提回归单元（Logistics Regression Unit）。人工神经元模型如图 1-2所示，这种结构又称为 McCulloch-Pitts 神经元。它将 n 个输入加权求和后，经过变换 $f(x)$输出。逻辑斯提回归单元的 $f()$函数就是逻辑斯提函数（Logistics Function）。

将这些单个人工神经元联网，形成复杂的人工神经网络结构，并可以不断扩大网络的层数（又称为深度）和人工神经元的数目，如图 1-3 所示。

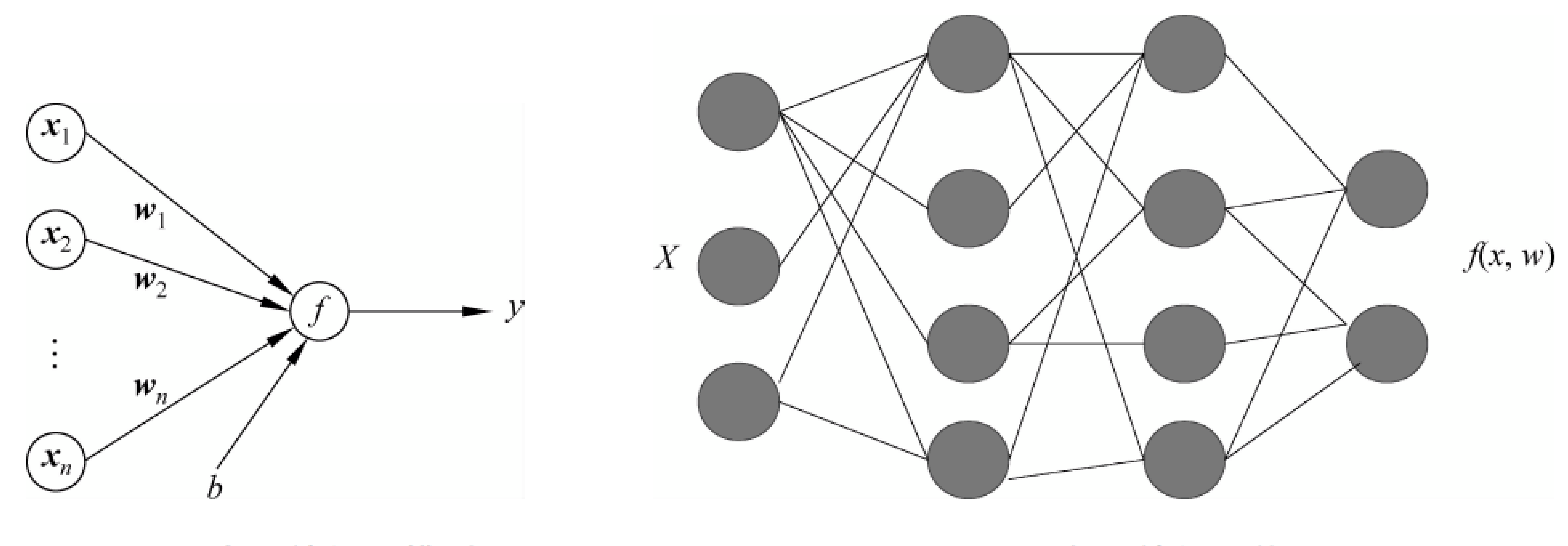


图 1-2 人工神经元模型　　图 1-3 多层神经网络

以上架构人工智能的神经网络方法，就是典型的连接主义的方法，现在称为深度学习的方法，在有大量的训练数据和超大计算能力的情况下，在工业实践中被证明是有效的。

这里需要注意：深度学习使用的神经网络，本质上是一个函数变换的数学模型，和生物中的神经元与神经网络已经没有什么关系了。

基于人工神经网络的人工智能技术的发展，并非一帆风顺。过去经历了两次高潮和

两次低谷。

随着 1957 年罗森布赖特提出了感知机(Perceptron)的概念,掀起了第一次人工神经网络的热潮。由于受当时计算机的计算能力限制,20 世纪 70 年代进入了低谷。

1986 年,随着霍普菲尔德神经网络与 BP 算法的提出,掀起了第二次人工神经网络的热潮。这次由于人工智能计算机的研制失败,20 世纪 90 年代再次进入了低谷。

2006 年,多伦多大学的 Geoffrey Hinton 提出了深度神经网络和训练方法。

2011 年,深度神经网络在 TIMIT 语音识别上实现了突破。

2013 年,深度神经网络在图像识别上取得了重大进展。

2013 年,深度学习被列为《麻省理工学院技术评论》的十大突破性技术之首。

目前,人工神经网络正进入第三次热潮。回顾过去,人们发现主要原因是当时的计算机的计算能力不够,用于训练的数据样本量不足,造成期望与实际效果之间有较大差别。随着计算硬件技术的进步,云计算和网络所支持的计算能力的大规模提升,再加上基于大数据的机器学习的算法进步,基于神经网络的人工智能方法得到复兴。

1.2.3 神经网络的复兴

深度学习(Deep Learning)是深度神经网络(Deep Neural Networks)的另一个名称。深度学习的核心是深度神经网络的设计与训练,采用层级更深、参数规模更大的神经网络。

深度神经网络的兴起在于三位领军人物的不懈追求。他们是 Geoffrey Hinton(多伦多大学教授)、Yoshua Bengio(蒙特利尔大学教授)和 Yann LeCun(纽约大学教授),见图 1-4。Geoffrey Hinton 参与提出了反向传播算法 BP,Yoshua Bengio 提出了 Autoencoder 和机



(a) Geoffrey Hinton

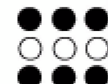


(b) Yoshua Bengio



(c) Yann LeCun

图 1-4　三位领军人物



器翻译的 GRU 等方法，Yann LeCun 提出了用卷积网络识别手写体的方法。

这次深度学习普及的引发点始于 2012 年，由 Geoffrey Hinton 指导博士生 Alex Krizhevsky 和 Ilya Sutskever 采用深度卷积网络(AlexNet)，在 ILSVRC-2012 图像分类(Image Classification)挑战赛的突破性的成绩，使准确率大幅度提升。

1.3 机器学习

1.3.1 机器学习的基本原理

深度神经网络是机器学习(Machine Learning)的一个分支。为了深入理解深度学习，我们有必要对机器学习的背景进行介绍。

机器学习的一个基本定义：给定一个计算机任务 T 和一个对任务 T 的性能度量 P，在给出经验集 E 的前提下，计算机任务 T 在性能度量 P 上有所提升。这个利用经验集 E 提升任务 T 的性能 P 的方法就是机器学习。

一般机器学习的原理如图 1-5 所示。机器学习是用数据训练模型，用模型进行预测，根据反馈产生数据，更新模型和数据。所以，机器学习包括数据、模型与算法 3 个方面。

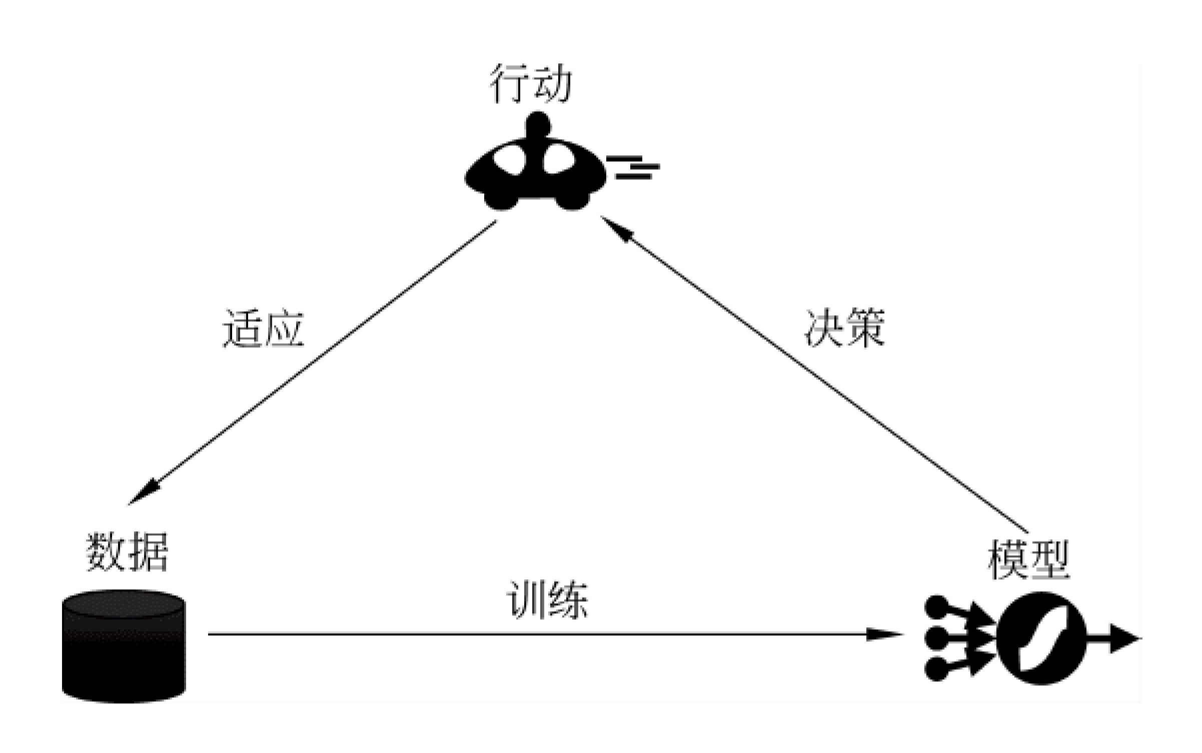


图 1-5 机器学习的原理

自 2012 年以来，基于深度学习的图像分类方法 AlexNet 在 ILSVRC 2012 比赛中的突破性表现，引起了各方关注，使人工智能得到新的发展。

在过去的几年里，深度学习在解决语音识别与图像处理等机器感知问题方面，表现优越，甚至超过人类的水平。目前，深度学习还在尝试解决自然语言理解、推理、注意和

记忆(RAM)等机器认知相关的问题。

现在的业界认为实现通用人工智能(强人工智能)的一种途径是深度学习和深度增强学习。

1.3.2 机器学习泛化能力

广义上讲,机器学习的成功依赖于它的泛化能力(Generalization)。通过在训练数据上的学习,然后能够推广到新的数据集上的能力称为泛化。

泛化后与正确的分类结果产生的误差称为泛化误差(Generalization Error,GE)。用数学公式表示为

$$GE = AE + EE + OE$$

其中,逼近误差(Approximation Error,AE)是指由于模型规模方面而产生的误差,要想减少这部分误差,需要扩大模型规模。

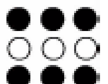
估计误差(Estimation Error,EE)是指由于数据集规模而产生的误差,要想减少这部分误差,需要增加可用数据的规模。

优化误差(Optimization Error,OE)是指由于算法设计而产生的误差,要降低这部分误差,需要设计更优的算法。

1.3.3 大数据是深度学习的基础

传统机器学习方法主要涉及数据、模型和算法 3 个方面。传统机器学习方法多采用手工或人为的特征选取,随着训练数据规模的提高,这种方法的提升效果就不明显了。而以深度学习为代表的方法,随着训练数据规模的扩大,提升效果显著,大大超过了传统机器学习方法。这种差异在语音识别、图像分类等机器感知类的任务上的体现尤其显著,如图 1-6 所示。

神经网络通过扩展网络结构的深度,扩大规模,甚至可以不断扩展下去,而且扩展之后所带来的效果是稳步提升的。通过规模的扩展,或者是提升网络的深度,是改进深度学习效果的途径。



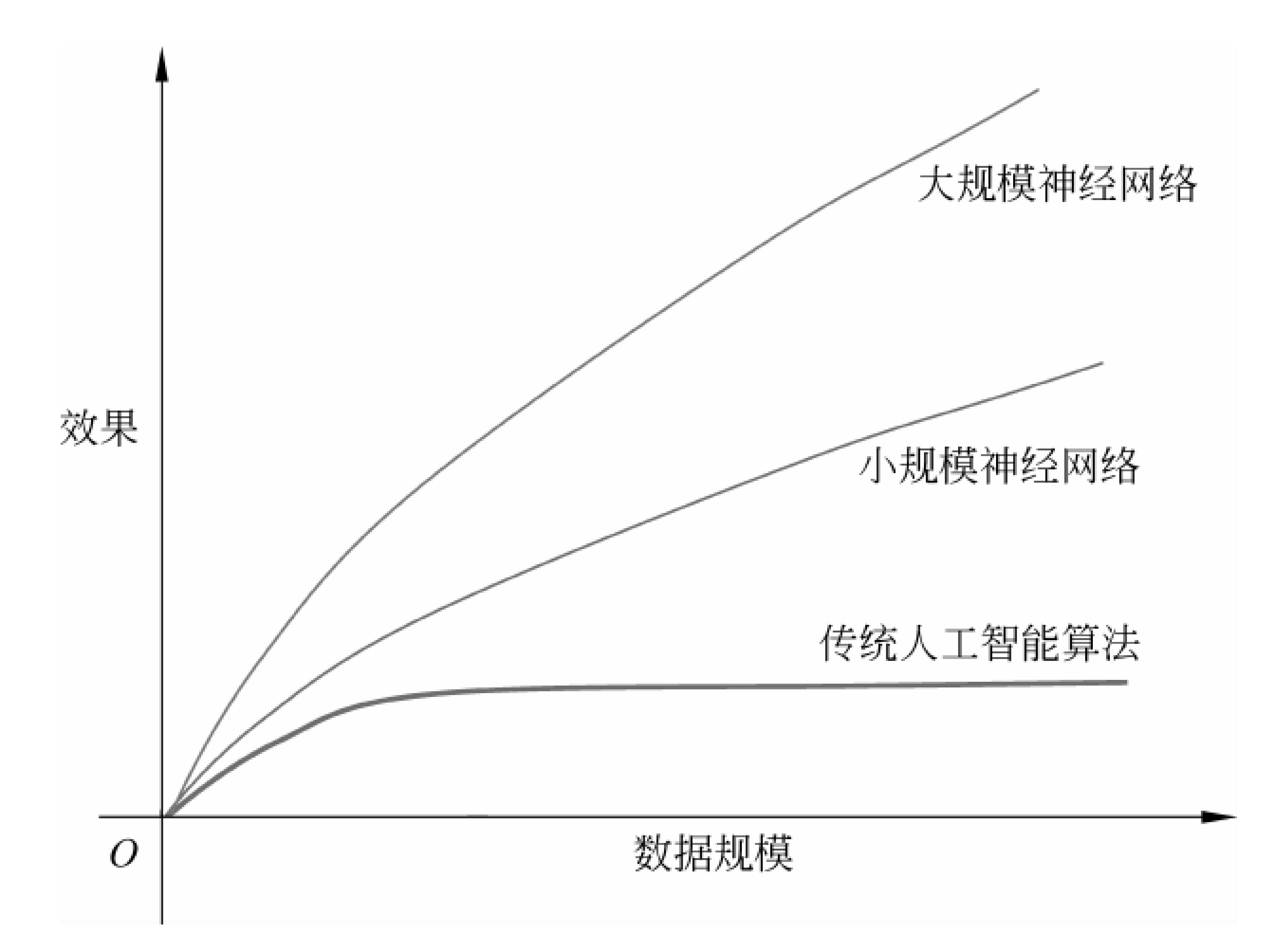


图 1-6　机器学习效果与数据规模之间的关系

参考文献

[1] Human B[EB/OL]. https://en.wikipedia.org/wiki/Brain.

[2] Deep Learning[EB/OL]. https://en.wikipedia.org/wiki/Deep_learning.

[3] Li Deng. Deep Learning for AI-from Machine Perception to Machine Cognition[C]. Keynote ICASSP 2016.

[4] Krizhevsky A, Sutskever I, Hinton G E. ImageNet Classification with Deep Convolutional Neural Networks[C]//International Conference on Neural Information Processing Systems. Curran Associates Inc. 2012:1097-1105.

[5] Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with Deep Neural Networks and Tree Search[J]. Nature, 2016, 529(7587):484.

[6] Temam O. The Rebirth of Neural Networks[C]//International Symposium on Computer Architecture. ACM, 2010:349-349.

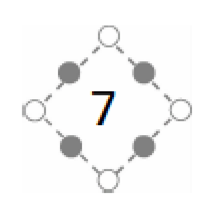
[7] Russakovsky O, Deng J, Su H, et al. ImageNet Large Scale Visual Recognition Challenge[J]. International Journal of Computer Vision, 2015, 115(3):211-252.

[8] 李航. 统计学习方法[M]. 北京：清华大学出版社，2012.

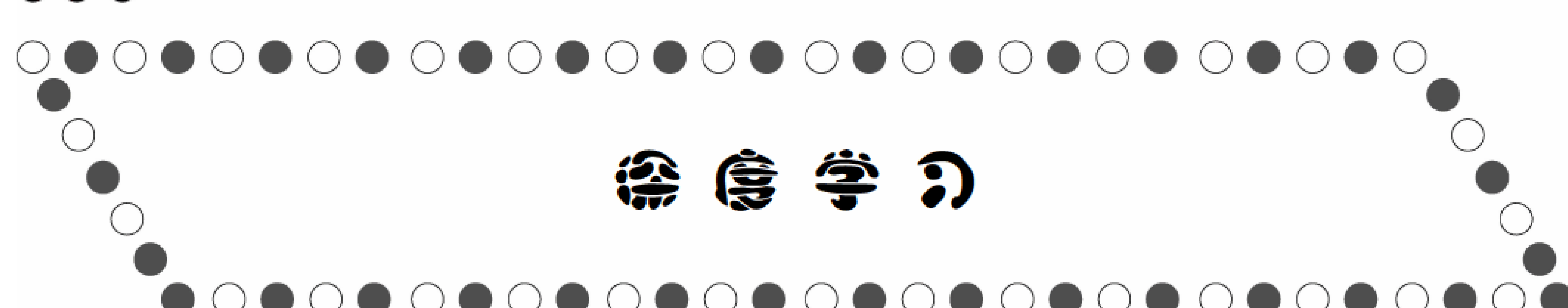
[9] 周志华. 机器学习[M]. 北京：清华大学出版社，2016.

[10] McCulloch W S, Pitts W. A Logical Calculus of Ideas Immanent in Nervous Activity[J]. Bulletin of Mathematical Biophysics, 1943,5(13):115-133.

[11] Ian Goodfellow, et al. Deep learning[M]. Cambridge: The MIT Press, 2016.







2.1 深度学习的原理

深度学习(Deep Learning)是深度神经网络(Deep Neural Networks)的另一个名称。深度学习的核心是深度神经网络的设计与训练,采用层级更深、参数规模更大的神经网络。

神经网络可以看作是对人脑的智能原理的一个数学抽象,包括神经元的结构和神经元的连接结构。其中,神经元的一个建模是逻辑回归单元。

深度学习通过单一的算法学习特征表示。逻辑回归是最常用的分类算法,也是深度学习的基本组成单元。

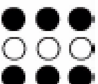
2.1.1 人工神经元

人工神经元可以理解为一组输入加权叠加,再经过一个连续可导的非线性变换进行输出。为了叙述方便,以下人工神经元简称神经元。

1. 逻辑斯提回归单元

逻辑斯提回归单元(Logistic Regression Unit)是最简单的人工神经元结构。单个逻辑斯提回归单元可以进行二类分类,多个逻辑斯提回归单元的组合,就可以完成复杂的分类工作。多层逻辑斯提回归单元在实际中表现出更好的性能。

单个逻辑斯提回归单元的数学表达式如下:



$$y = f(\sum_{i=1}^{n} w_i x_i + b)$$

其中，w_i 与 x_i 分别为权重与输入，y 为输出结果。$f()$函数采用逻辑斯提函数或逻辑曲线（Logistic Function）。$f()$函数的数学表达式和形状（见图 2-1）如下：

$$f(x) = \frac{1}{1 + e^{-x}}$$

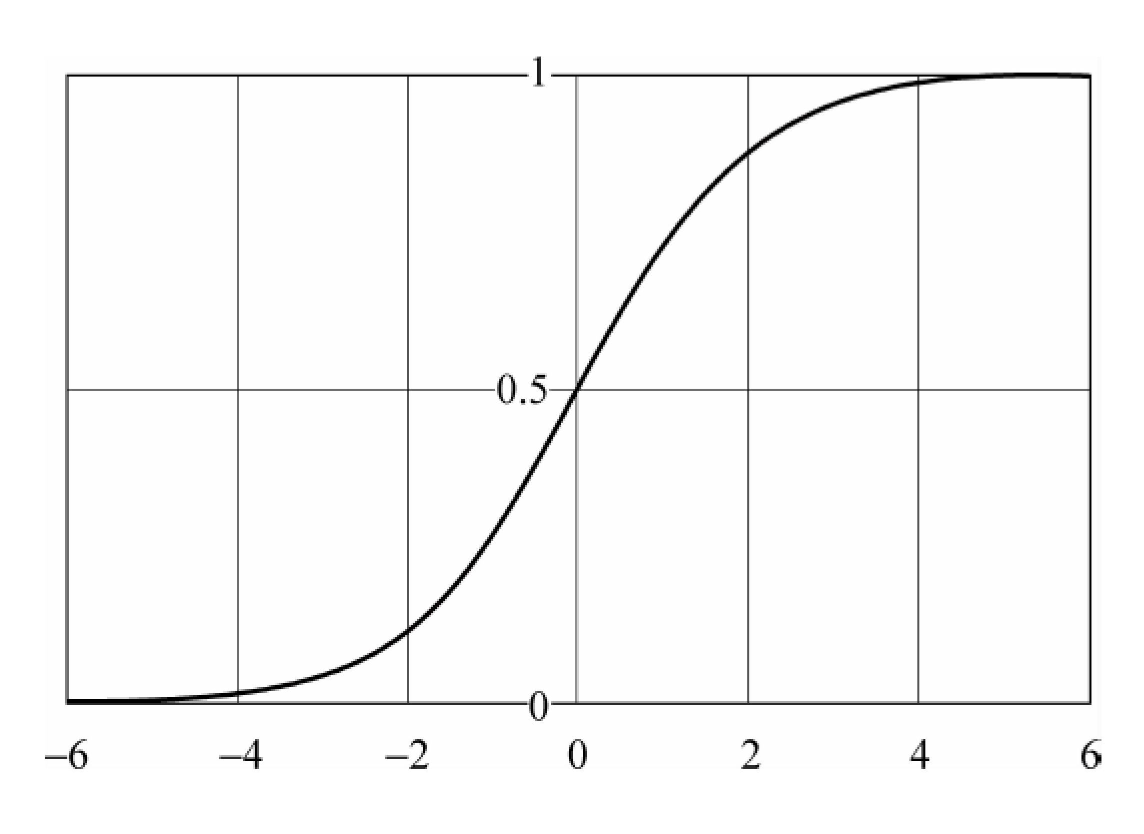


图 2-1　$f()$函数的形状

其图形是一条 S 形曲线（Sigmoid Curve）。逻辑函数因其曲线形状为 S 形，所以，也被称为 S()函数 或 Sigmoid()函数。Sigmoid()函数的取值为(0,1)。

2. 激活函数

$f()$函数也称为人工神经元的激活函数（Activation Function or Trigger Function）。当 $f()$函数采用逻辑斯提函数时，则人工神经元就是逻辑斯提回归单元。除了逻辑斯提回归单元的形式，人工神经元还有其他形式。例如，$f()$函数可以是 tanh()函数、Softmax()函数和 ReLU()函数（Rectified Linear Units）等。

双曲正切函数 tanh()，其数学表达式如下：

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

tanh()函数和 Sigmoid()函数是同类的，只是取值变为(−1,1)。

ReLU()函数的数学表达式如下：

$$\mathrm{ReLU}(x) = \max(x, 0)$$

ReLU()还有其推广形式，称为PReLU(Parametric ReLU)。PReLU()的数学表达式如下，其中 $\alpha \approx 0.01$：

$$\mathrm{PReLU}(x)=\begin{cases}\alpha x & \text{如果 } x<0\\ x & \text{如果 } x\geqslant 0\end{cases}$$

Softmax()函数(又称为归一化指数函数)多用于输出一个识别对象的概率分布。Softmax()函数的数学表达式如下：

$$g(z_m)=\frac{\mathrm{e}^{z_m}}{\sum\limits_{k}\mathrm{e}^{z_k}}$$

2.1.2 多层人工神经网络

将多个人工神经元排列连接起来，就形成了多层人工神经网络。图 2-2 给出了一个二层神经网络结构的例子。

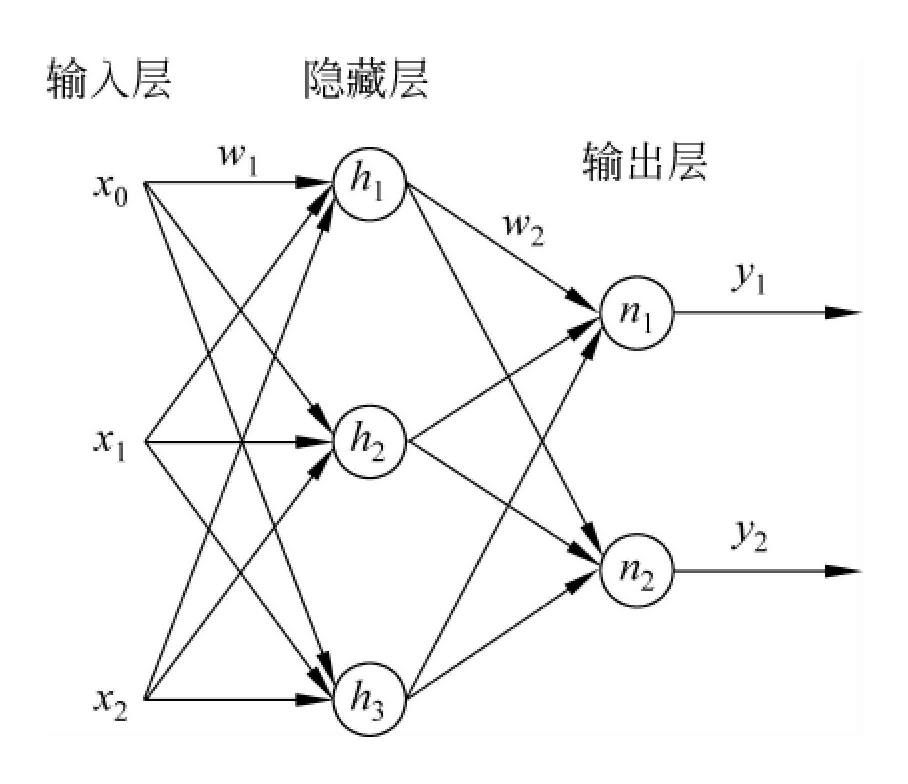
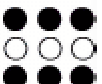


图 2-2 二层神经网络

该神经网络分为输入层、隐藏层和输出层。输入层有 3 个输入(x_0，x_1，x_2)，中间有一个隐藏层，输出层有 2 个输出(y_1，y_2)，共有 5 个神经元。

该网络的连接结构采用多层全连接(Fully Connected Network，FCN)的模式，是人工神经网络中最简单的连接方式。

为了叙述方便，以下人工神经网络简称神经网络。



2.1.3 神经网络训练

如果激活函数 $f(x)$是一个连续可导函数，如 Sigmoid()函数，那么神经网络本质上是一个由输入和内部权重作为变量的连续可导函数。

神经网络的训练过程属于机器学习的监督学习(Supervise Learning)，即训练数据上都有相应的标签(Label)。神经网络对具有标签的训练样本进行学习，从而确定网络的权重参数，然后用训练得到网络，对训练样本集外的数据进行标签的预测。

神经网络训练的本质，就是找到相应的内部权重，使得在训练数据(样本)输入到网络后，网络的实际输出与预期输出(即标签)之间差异最小。

1. 损失函数最小化

神经网络的输出结果会与实际所对应的标签之间存在差别，可以引入相似度度量函数，来表示这种差异，称为损失函数(Loss())、成本函数(Cost())或代价函数。

神经网络训练的目标是使损失函数最小，所以，训练过程是损失函数的求最小化过程。损失函数可以通过交叉熵的方式，来计算损失。其他的损失函数形式还有均方求和、均方根等度量函数，但是这类函数作为损失函数的缺点是它们属于非凸函数，存在很多极小值点。

交叉熵形式的损失函数表示如下：

$$H_{y'}(y)=-\sum_i y'_i \ln y_i$$

其中，y'表示训练样本对应的标签，y 表示神经网络的输出。

实际中还会在损失函数中加上一些正则项(Regularizer)或惩罚项(Penalty Term)，该过程称为正则化(Regularization)。目的是为了防止过学习(Over-fitting)。常见的惩罚项是 $\alpha\sum_j\sum_i\|W_{ij}\|$。

2. 反向传播算法

这里给出多层神经网络的一种更加抽象表示，这种表示抽象了每层的基本特征，如图 2-3 所示。

根据损失函数的性质和链式求导法则，可以反向逐层计算损失函数对权重的导数，调整权重最小化损失函数。这个方法称为反向传播算法(Back-Propagation Algorithm)。多层神经网络的反向传播的过程如图 2-4 所示。

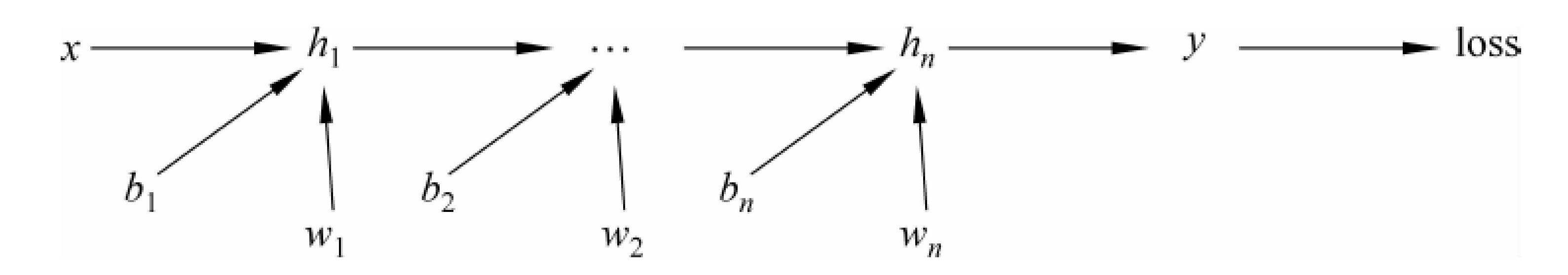


图 2-3　多层神经网络

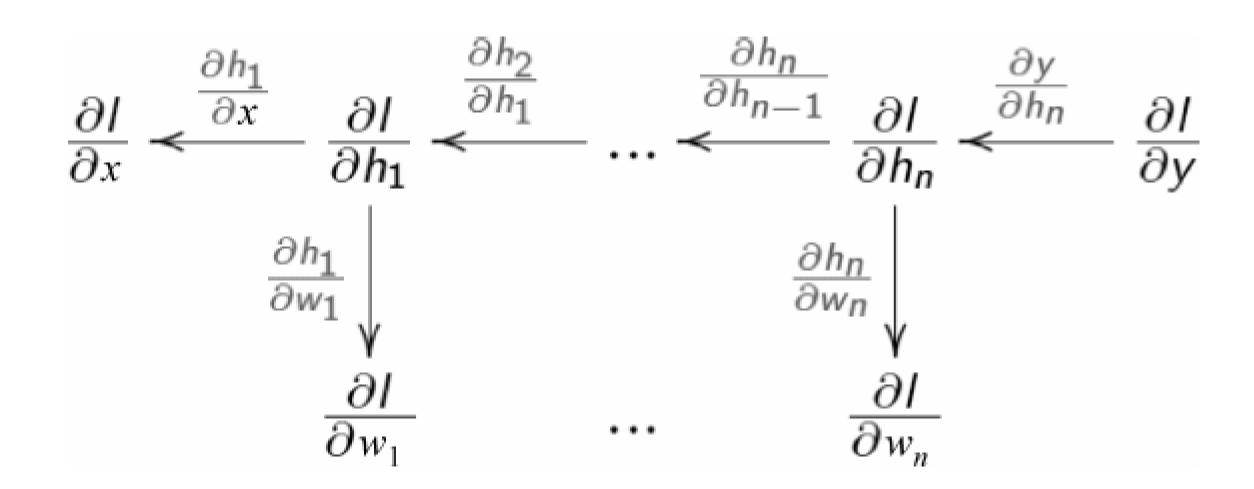


图 2-4　多层神经网络的反向传播方法

3. 随机梯度下降法

随机梯度下降方法(Stochastic Gradient Descent，SGD)是最常用的权重调节方法，基本方法如下。

步骤 1：随机初始化每个神经元的输入权重和偏差(Weights and Bias)。

步骤 2：选取一个随机样本(Samples)。

步骤 3：根据神经网络的输出结果，从最后一层开始，逐层计算每层权重的偏导数。

步骤 4：逐层调整每层的权重，产生新的权重值。

返回到步骤 2，继续随机选取下一个样本。

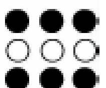
4. 实际训练过程

多层神经网络的实际训练过程是将样本数据“分批训练”，同时采用随机梯度下降法和反向传播方法，逐层调整权重参数。

训练过程中的一些术语解释如下。

(1) 先将整个训练集分成多个同样大小的子集，每个子集称为一个批次(Batch)，子集的大小(即样本数目)由参数批次大小(Batch Size)控制。

(2) 使用随机梯度下降法，其中每一步可以使用不止一个样本，这称为迷你批次(Minibatch)。每次迭代所用的样本数目称为迷你批次大小(Minibatch Size)。当迷你批次大小为 1 时，就是普通的随机梯度下降。



(3) 每个批次的数据被依次送入网络进行训练，训练完一个批次，被称为一次迭代(Iteration)。

(4) 训练集的所有训练样本都被送入网络训练，完成一次训练的过程，称为一个时代(Epoch)。

(5) 时代记录了整个训练集被反复训练的次数，而迭代记录网络权重参数的调整次数。

迭代和时代之间的数量关系由批次大小和训练集大小决定。

5. 自编码器

1) 自编码器初始化权重

自编码器(Autoencoder)主要用于多层神经网络参数的初始化。采用随机化网络参数的方法，会导致网络收敛慢的问题。而采用自编码器方法，在一些训练过程中证明是有效的。自编码器的结构如图 2-5 所示。

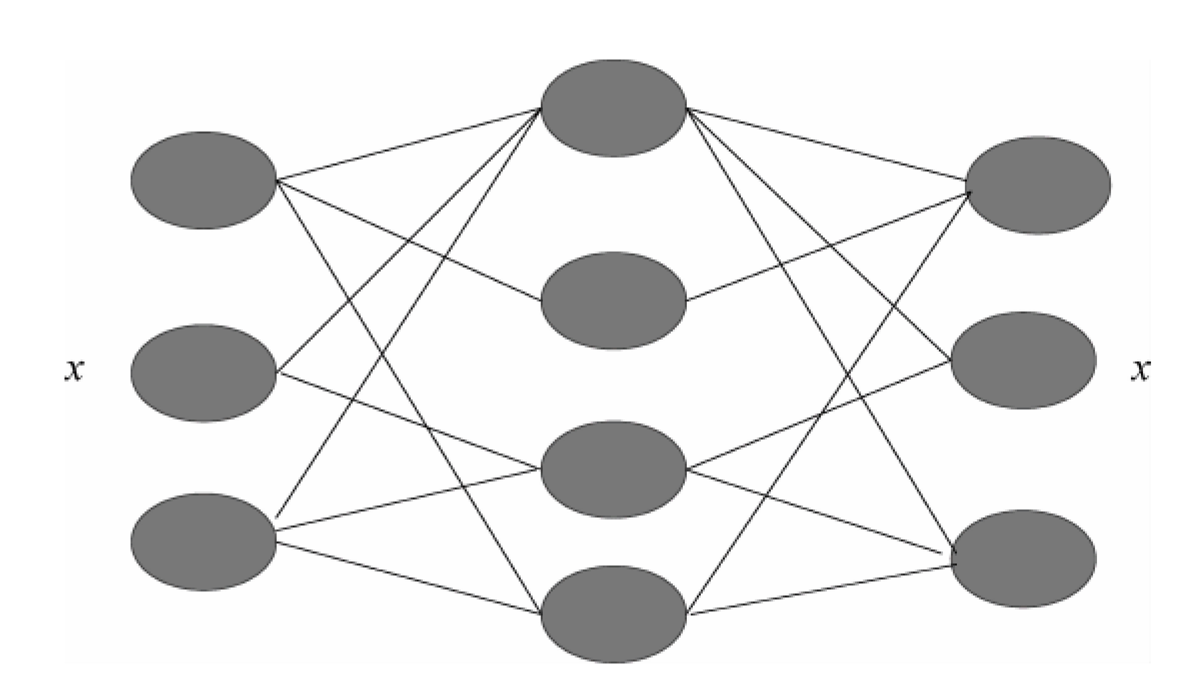


图 2-5 自编码器的结构

自编码器的功能是神经网络的输出尽可能地还原输入的特征。这样可以对多层神经网络的每一层逐层去构造自编码器，用训练出来的权重作为多层神经网络每一层的初始权重。

自编码器初始化多层神经网络的参数方法描述如下。

(1) 预训练：使用无监督的数据，对每一层进行预训练。

(2) 微调步骤 1：对最后的输出层用监督学习的标记数据，进行训练。

(3) 微调步骤 2：使用标记数据，对整个网络进行反向传播训练。

2）降噪自编码器

降噪自编码器(Denoising Autoencoder,DA)的原理是人工破坏或者屏蔽部分输入，输入到自编码器进行训练，让自编码器学到数据间的关联性。实际操作中，屏蔽是随机进行的，所以，神经网络最终学到的是通过部分数据联想到缺失数据的能力。

举个例子，输入一个9维的数据，通过屏蔽最后一维，比如将第9维数据(置零)，得到一个8维+0的9维数据。将此数据经过一个9－3－9的神经网络处理，得到新的9维数据。训练要求新的9维数据和没被屏蔽前的输入数据一样。这样，自编码器至少完成了这样一件事：给定前8维，推测最后一维。

这种推测不是简单的“线性插值”，因为神经网络的每层输出，都要通过激活函数，进行了一次非线性函数变化。所以，这种推测肯定不是线性插值的，而是一种更复杂的数值插值方法。

具体参考链接 http://deeplearning.net/tutorial/dA.html。

6. 训练的小技巧

这些神经网络训练调试的小技巧源于 Le Quoc V 的教程。因为当时还没有完整成熟的深度学习框架，很多情况下需要自己去调整优化。有了成熟的框架后，有些问题就不用太关心了。

(1) 用数值近似的方法检查反向传播算法的梯度计算的正确性(依赖选择的框架)。

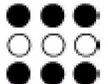
(2) 随机初始化参数是很重要的。一个好的方法是参数初始化采用高斯分布或均匀分布。有时调整一下初始化的协方差又会有帮助(依赖选择的框架)。

(3) 确保随机初始化不要“饱和”网络。这意味着绝大多数时间，神经元的输出为0.2～0.8。如果神经元输出太多的0或1，梯度值会比较小，训练时间会比较长。

(4) 能够有效地监控训练过程(依赖选择的框架)。

(5) 选取好的学习率。建议学习率为0.1或0.01。学习率过大，参数值改变过于激进；学习率过小，参数值改变过于保守。

(6) 选取好的超参数(如层数、每层神经元数目)是一个当前研究的问题。一种好的方法是采用交叉验证(Cross Validation)。选取一个验证集，与训练集完全无关。如果超参数在训练集上表现好，而在验证集上表现不佳，这个模型就是过拟合(Overfits)：模型有太多的自由度并记住了训练集的特征，但是无法推广。如果模型的超参数在训练集上表现极差，这就是欠拟合(Underfits)：模型没有足够的自由度，应该增加隐藏层数或者



神经元的数目。另外,在训练集上表现不佳,也可能是学习率选取不好(比较重要)。

(7) 选取好的超参数,也可以使用 Gridsearch 方法,随机搜索或贝叶斯优化。在 Gridsearch 方法中,所有的组合都会尝试,并用验证集进行交叉检验。如果 Gridsearch 成本太高,可以用随机搜索来产生配置。贝叶斯优化检查先前的超参数的网络组合,找出一个拟合函数,然后选择一个最大化效用函数的超参数组合。

(8) 神经网络需要长时间训练,需要花费时间去优化代码速度。例如,用快速矩阵矢量库、后向传播算法的矢量化版本(依赖选择的框架)。

(9) 可以采用单精度而不是用双精度来存放参数。这可以减少一半的存储空间,而不会损害神经网络的性能。缺点是用数值近似做梯度正确性的检查会复杂些。

(10) 神经网络的一个不方便之处是目标函数一般不是非凸函数。这意味着我们获得的最小值,很可能是局部最小而非全局最小。所以,神经网络对随机初始化比较敏感。学习过程的其他随机化方面也会影响结果。例如,学习率的选择,迭代样本的不同顺序会产生不同的优化参数。

(11) 在许多情况中,使用较大的迷你批次是一个好方法,因为可以降低样本的噪声(计算多个样本的平均值的例子)。另外,速度更快,因为矩阵向量库用更大的矩阵工作会更好。

2.2 典型的神经网络架构

经过不断的实验测试,人们发现不同的神经网络架构适合处理不同的任务。卷积神经网络适用于图像处理,因为这种网络结构适合图像处理的不变性,例如猫的图像,平移旋转处理后依然是猫。循环神经网络适合进行序列预测的任务,如语音识别、诗词填空等语音语言的处理。因为长短期记忆循环网络训练快,已经在语音识别中大规模应用。

2.2.1 卷积神经网络

对于高维数据如图像(200×200 像素),如果第一层是全连接的话,这意味着第一个隐藏层的每个神经元都有 40 000 个输入权重。为了效率更高,可以强制神经元的输入权重的数量,强制输入只和局部的神经元有连接。这样形成的网络又称为局部网络,或者局部连接网络。

权重共享(Weight Sharing)是一种简单的局部网络形成方法，如图 2-6 所示。这种权重共享等效于信号处理领域中的卷积运算(Convolution)。

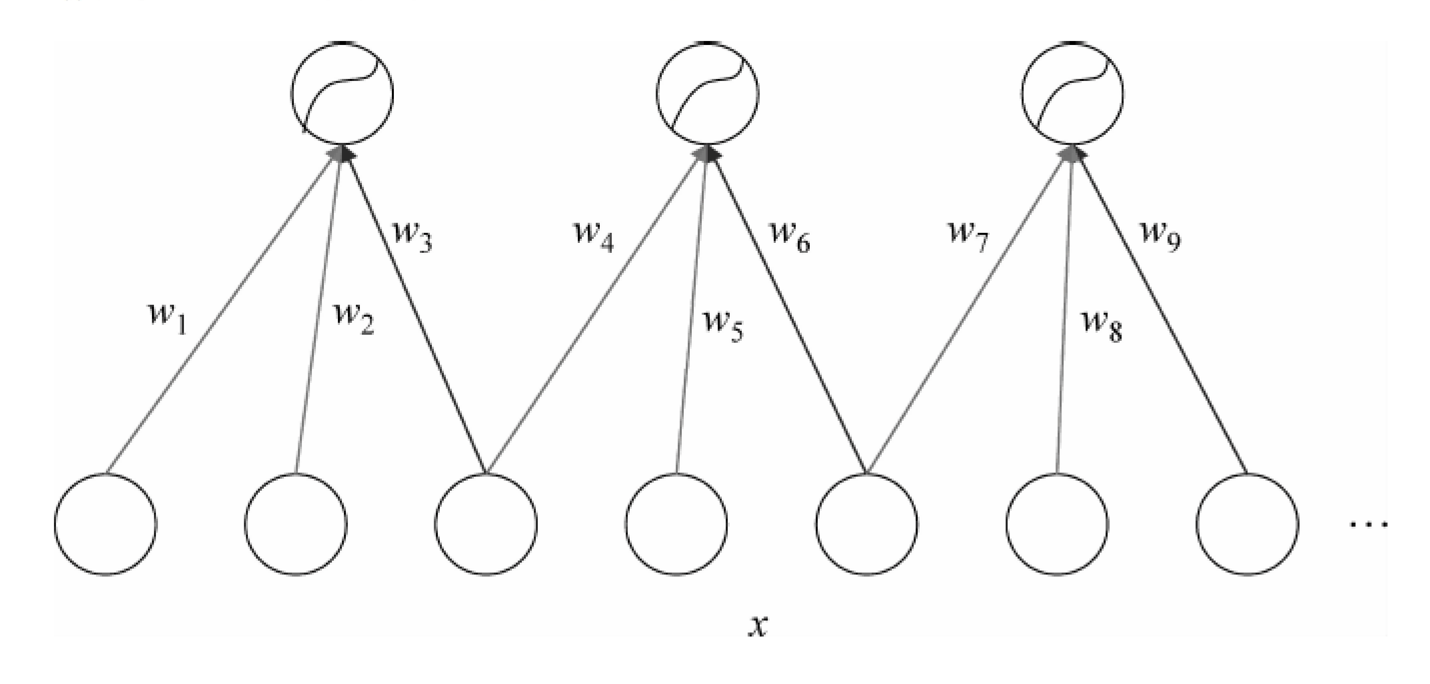


图 2-6　权重共享

实际中，每一个卷积层后紧跟着一个下采样层，例如采用最大池化(Max-pooling)方法完成下采样。最大池化层的操作，就是在过滤器之后(卷积之后)，计算出卷积层神经元输出的最大值。最大池化的操作原理如图 2-7 所示。

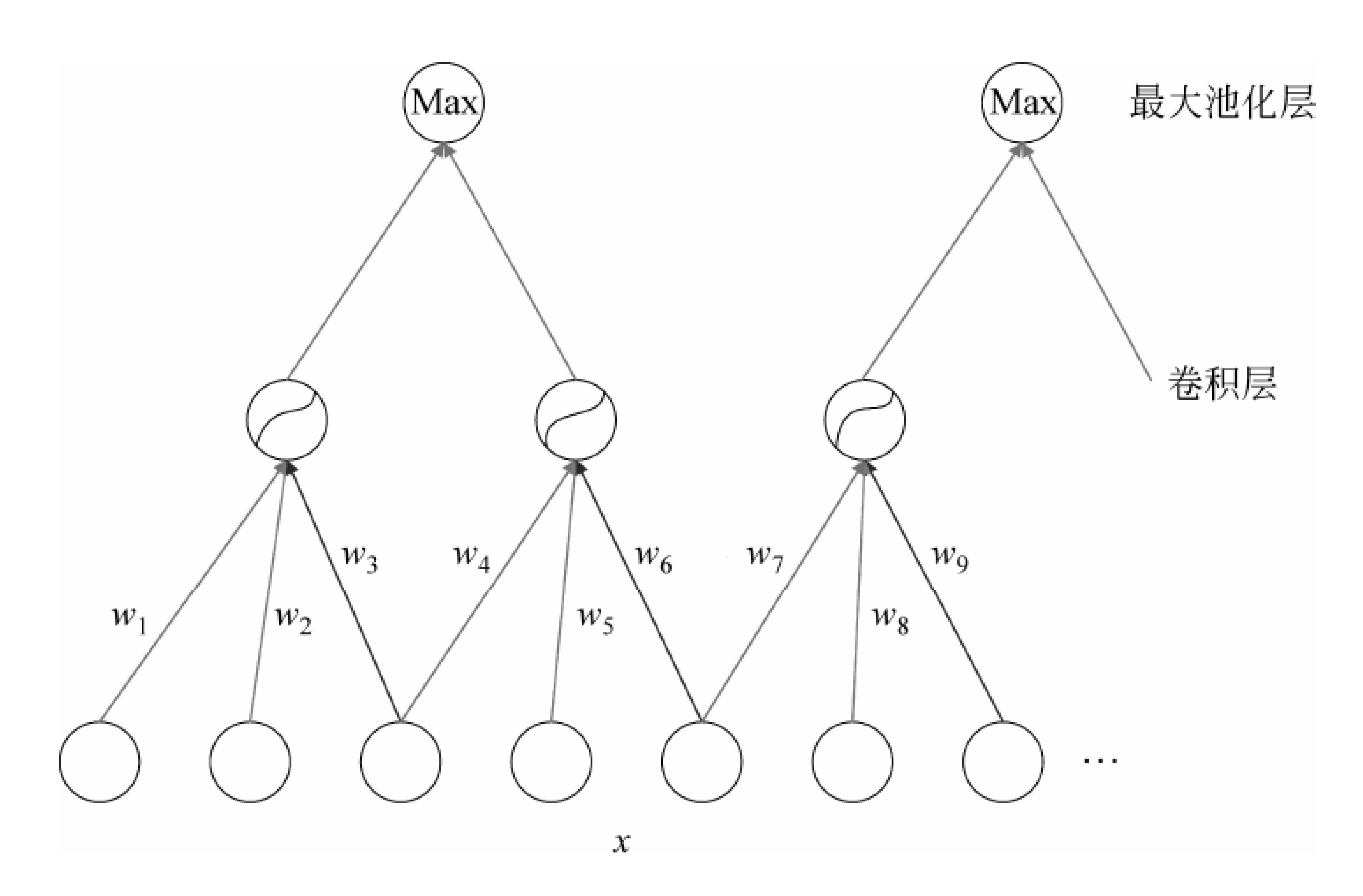
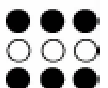


图 2-7　最大池化操作

卷积操作就是在输入信号上使用过滤器(Filter)，而过滤器的参数就是一组权重值。所以，这种局部区域的权重共享的神经网络，又称为卷积神经网络(Convolutional Neural Network，CNN)。一个卷积神经网络的工作原理如图 2-8 所示。



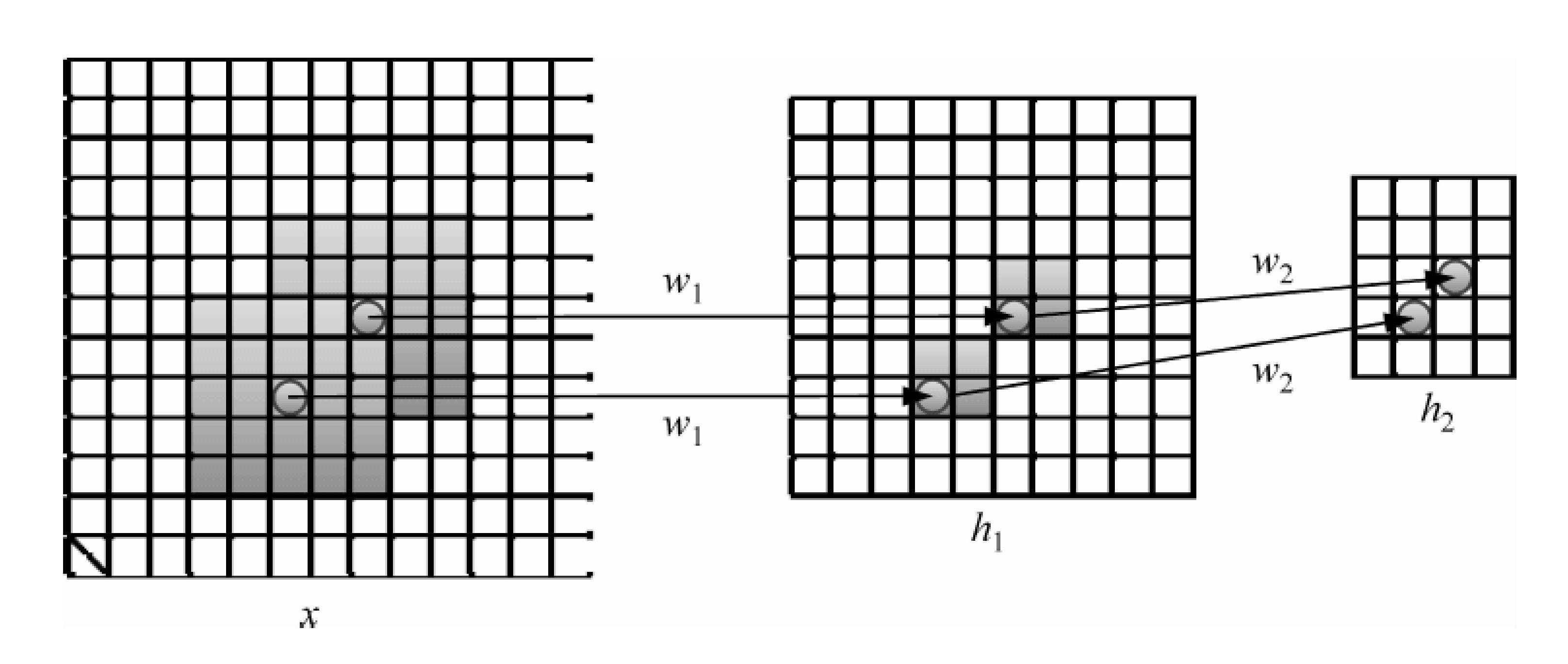


图 2-8　卷积神经网络

最新 CNN 进展还包括 LCN (Local Contrast Normalization)。LCN 操作在最大池化层之后，其目标是减去平均值，除以标准差。这个操作允许亮度不变性，对于图像识别用处很大。

2.2.2　循环神经网络

对于时间序列的预测是一个重要问题。循环神经网络(Recurrent Neural Network，RNN)的每一个时间步处理时，权重共享。一个单向 RNN 如图 2-9 所示。

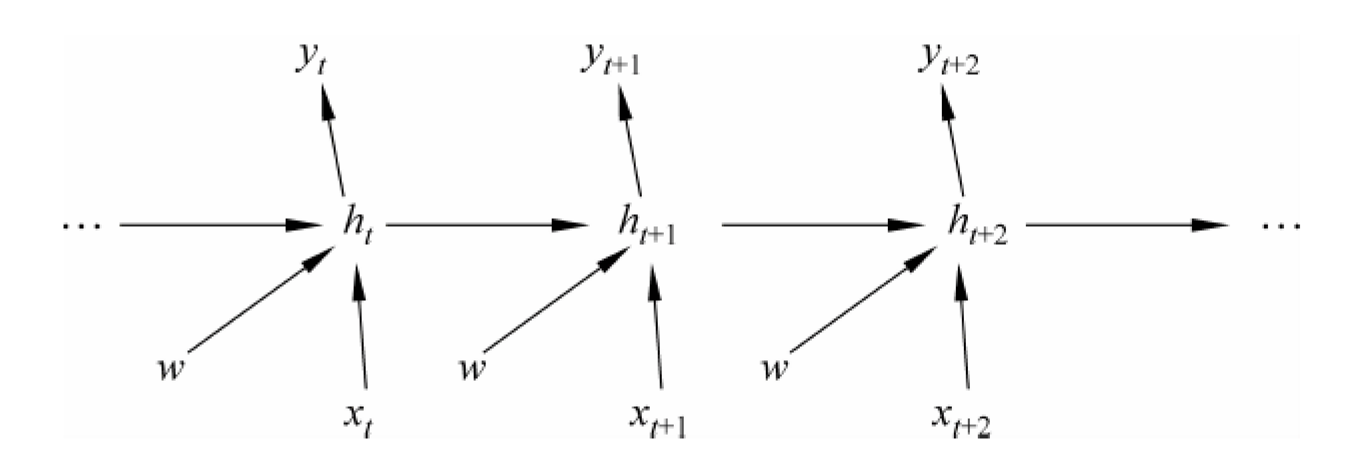


图 2-9　单向 RNN

RNN 会在时间域上进行反馈操作，具体来说，有两种反馈方式：一种是上一时刻或前面若干时刻的输出会被作为输入的一部分与真正的输入一起构成总的输入量，即前向方式；另一种是后面若干时刻的输出会被作为输入的一部分与真正的输入一起构成总输入量，即后向方式。如果前向方式和后向方式都存在，那么就被称为双向模式。一个双向 RNN 如图 2-10 所示。

双向网络这样的做法，就是采用两个单向网络结构，这样做增加了复杂性，但是充分利用了输入的前后相关性，识别的准确率也提高了。单向 RNN 的优点是结构简单，可以

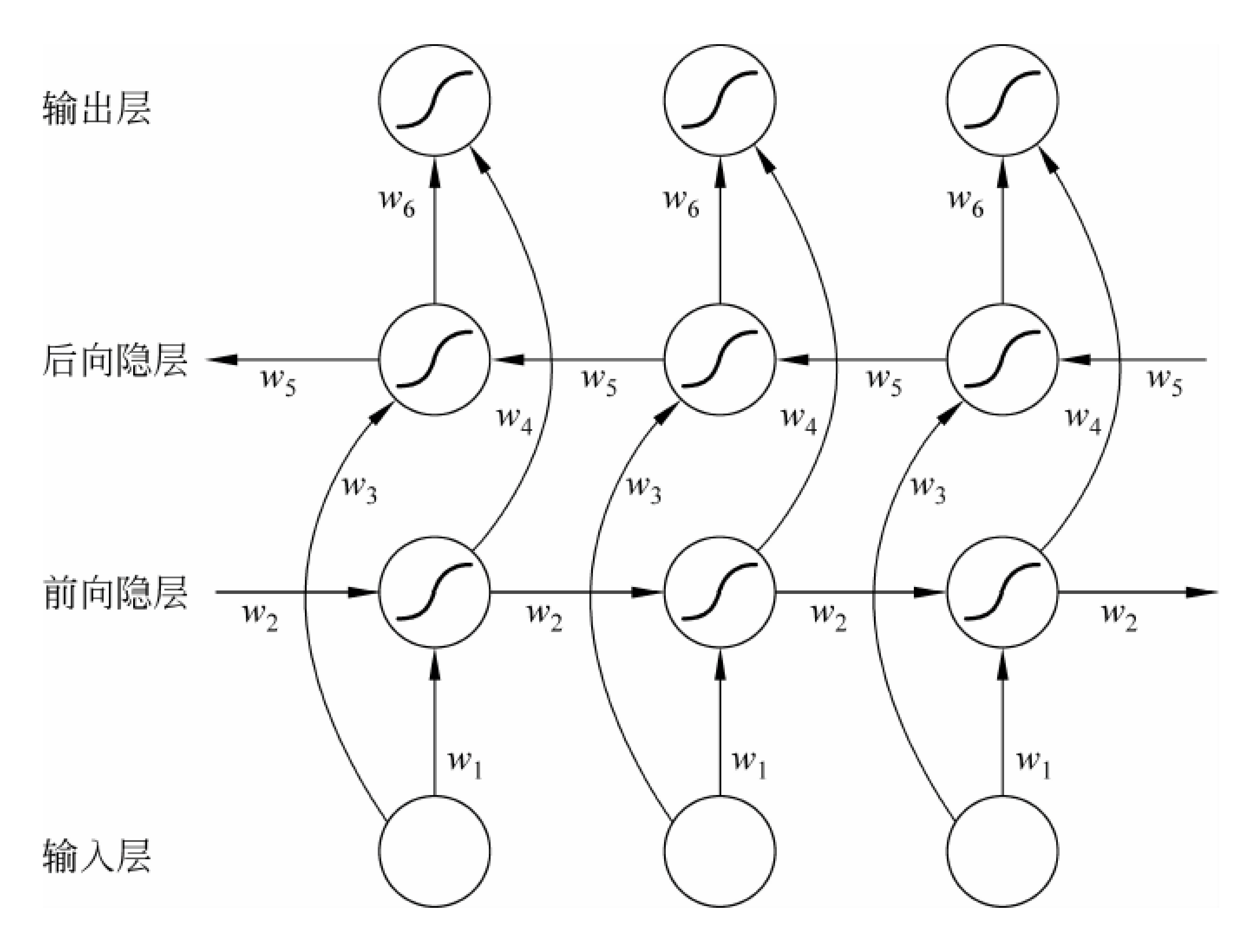


图 2-10　双向 RNN

完成实时的处理功能。

2.2.3　长短时记忆循环网络

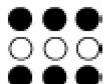
长短时记忆循环网络(Long-Short Term Memory Recurrent Network,LSTM)是RNN 的一个改进。相比于 RNN,LSTM 增加了一个主输入单元和其他 3 个辅助的门限输入单元:输入门(Input Gate)、记忆单元(Memory Cell)、遗忘门(Forget Gate)及输出门(Output Gate)。

输入门、遗忘门、输出门和记忆单元组合起来,大大提升了 RNN 处理远距离依赖问题的能力,解决了 RNN 的梯度值消失问题(Vanishing Gradient Problem),无法学习长程的相关性问题。

典型的 LSTM 结构是一个主输入单元和其他 3 个辅助的门限输入单元。3 个辅助的门限输入单元分别控制网络是否输入,是否存储输入以及是否输出。这样一来就可以寄存时间序列的输入。在训练过程中会利用后向传播的方式进行。

如图 2-11 所示,其中,输入门控制是否输入,遗忘门控制是否存储,输出门控制是否输出。

假设在 t 时刻的输入是 x_t,m_t 是m_{t-1} 和 f 的线性组合。



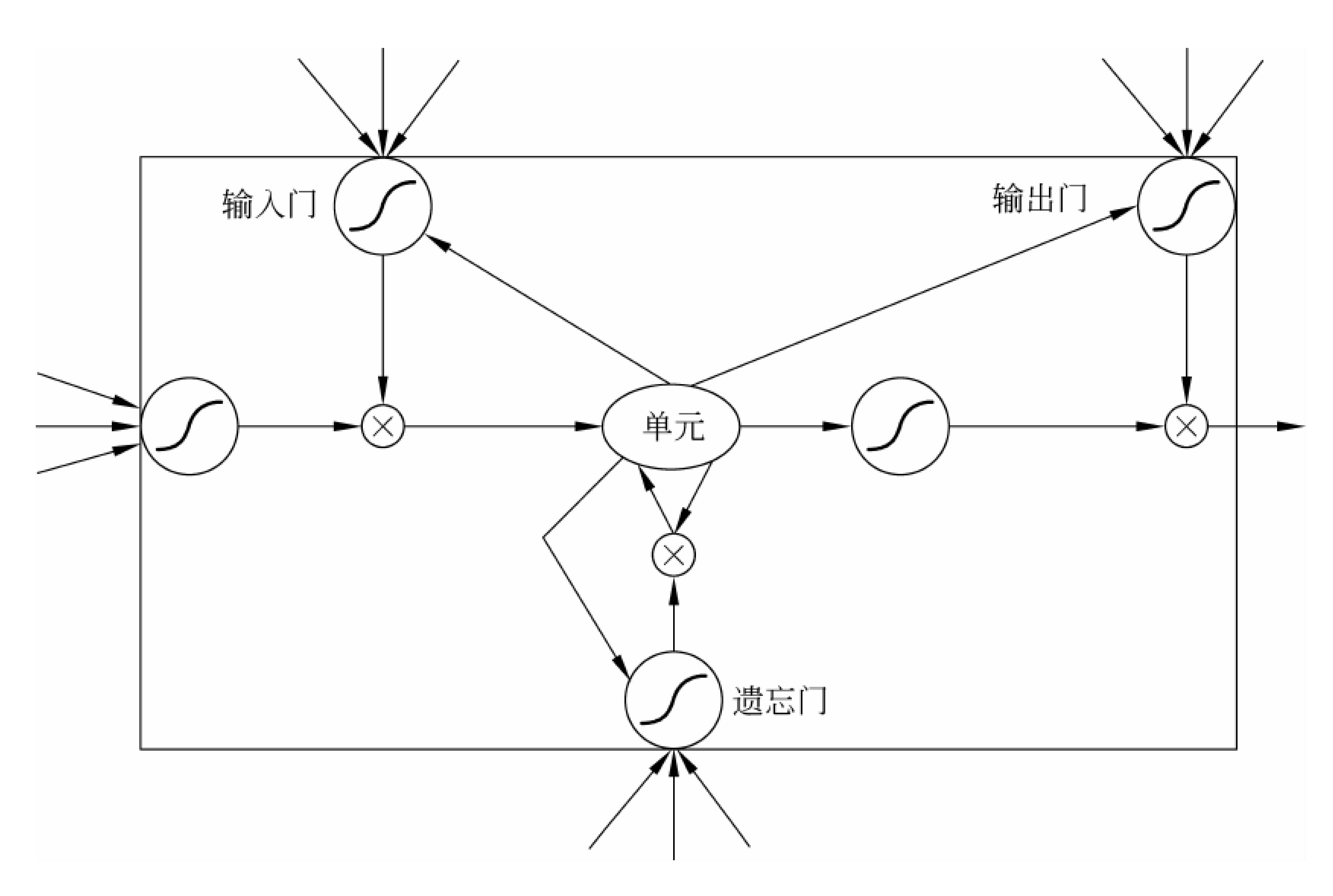


图 2-11 长短时记忆单元

$$m_{t-1} = \alpha \odot m_{t-1} + \beta \odot f(x_t, h_{t-1})$$

在 t 时刻的隐含层的状态 h_t，计算如下：

$$h_t = \gamma \odot \tanh(m_t)$$

其中，h_t 和 m_t 会在下一个时间步继续使用。α、β、γ 形式如下：

$$\alpha(t) = g_1(x_t, h_{t-1}, m_{t-1})$$

$$\beta(t) = g_2(x_t, h_{t-1}, m_{t-1})$$

$$\gamma(t) = g_3(x_t, h_{t-1}, m_t)$$

其中，α、β、γ 也被称为输入门、记忆门和输出门，分别控制了输入、记忆和输出到下一个时间。

LSTM 最早是由 Sepp Hochreiter 和 Jürgen Schmidhuber 提出。

2.2.4 门控循环单元循环网络

门控循环单元(Gated Recursive Unit，GRU)是由 Cho 等人在 LSTM 的基础上提出的简化版本，也是 RNN 的一种扩展。GRU 的结构如图 2-12 所示。

GRU 单元只有两个门，即重置门和更新门。

(1) 重置门(Reset Gate)：如果重置门关闭，会忽略掉历史信息，即历史不相干的信

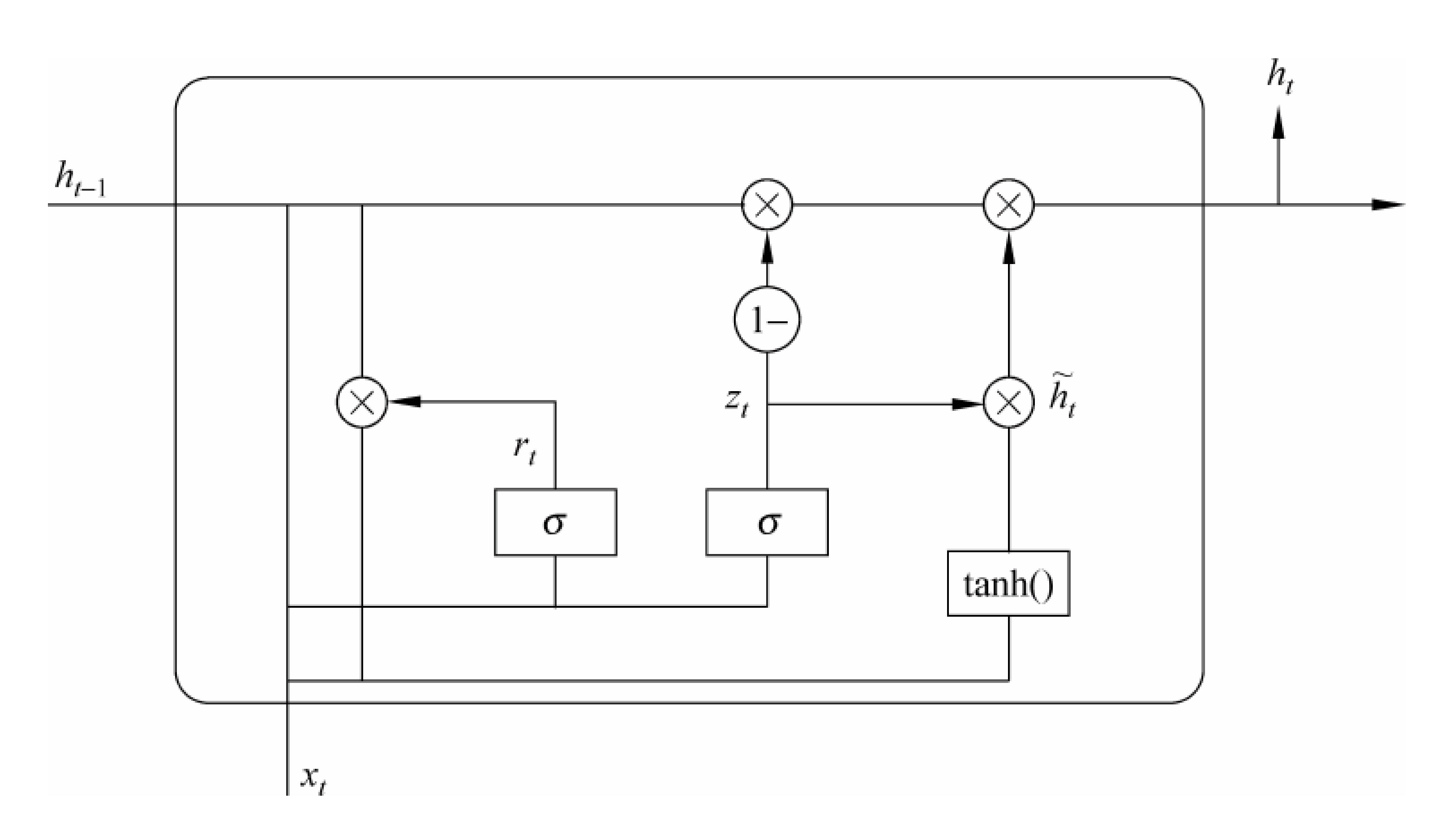


图 2-12　GRU 的结构

息不会影响未来的输出。重置门：

$$r_t = \sigma(W_t \cdot [h_{t-1}, x_t])$$

其中，σ 为 Sigmoid()函数。

(2) 更新门(Update Gate)：将 LSTM 的输入门和遗忘门合并，用于控制历史信息对当前时刻隐层输出的影响。如果更新门接近 1，会把历史信息传递下去。更新门：

$$z_t = \sigma(W_t \cdot [h_{t-1}, x_t])$$

其中，σ 为 Sigmoid()函数。

隐含层节点状态更新：

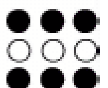
$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

其中，tanh()为双曲正切函数，输出值为(−1,1)。

输出：

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

GRU 论文最早是用于研究机器翻译(Machine Translation)，在语音识别中也被证明很有用。GRU 和 LSTM 在实用中，很多任务性能相当，但是 GRU 参数少，运算更简单也更快，需要的训练数据少。如果拥有的数据量足够多，LSTM 可以充分发挥表达能力强的优点。



2.3 机器感知

2.3.1 语音识别

1. 深度学习的语音识别

基于深度学习的语音识别的通用流程包括语音输入环节、深度神经网络的结构和辅助语言模型。

1）语音输入环节的处理

对于语音输入环节，通常有两种处理方式。

（1）基于语音的时频谱图（Spectrogram）。对于每一段语音，要通过傅里叶变换，将其每个时间段上的频谱展示出来，然后形成一个横轴为时间、纵轴为频率，以不同颜色区分对应频率幅度的频谱图。

（2）直接输入原始波形，然后对波形进行时间域上的卷积变换。也就是利用滤波器组（Filter Banks）提取不同频率上的信号，从而得到对应的时间-频率关系。

2）深度神经网络的结构

对于以语音的频谱特征为输入的神经网络而言，有 3 种通常的神经网络结构。一种是采用 CNN，即卷积神经网络的方法。另一种是利用 RNN，即循环神经网络的方法。RNN 的一种替换方式是 LSTM，即长短时记忆循环网络。LSTM 的一个简化版本是 GRU。最后一种是利用 FCN（Fully Connected Network）或 DNN，即全连接网络。

3）辅助语言模型

N-gram 语言模型（*N*-gram Language Models）主要是对前面的神经网络输出进行进一步的处理优化。*N*-gram 考虑了前后连续 *N* 个词语之间的相互联系，通过统计词频的方式，近似地计算前后若干词一起按序出现的概率，最终得出整个语句出现的概率。

2. 混合模型或合金模型

过去，利用高斯混合模型（Gaussian Mixture Model，GMM）以及隐藏马尔可夫模型（Hidden Markov Model，HMM）实现了大词汇量级下的连续语音识别（Large Vocabulary Continuous Speech Recognition，LVCSR）。传统使用 HMM 和 GMM 混合方案，其中，HMM 用来规范时间的变化，GMM 用来计算 HMM 之中各个组合的可

能性。

深度神经网络(DNN)可以与隐藏马尔可夫模型、上下文相关模型(Context-dependent Phone Models)、N-gram 语言模型和维特比搜索算法(Viterbi Search Algorithms)进行混合使用。

尽管可以采用混合方法来构建语音识别引擎。但是,混合模型一般比较复杂,需要一套精致的训练方法,以及相当多的专业知识来帮助搭建模型。

3. CTC 方法

CTC(Connectionist Temporal Classification)称为连接主义的时间分类。

CTC 最早是由 Alex Graves 等人提出,用来解决找到概率最高的标签序列,解决语音文件训练中的标记问题。CTC 模型描述如下。

输入:一个 m 维的时间序列输入 $x = \{x_1 \cdots x_t\}$,其中$x_t \in \mathbf{R}^m$。

输出:一个序列 $z = \{z_1 \cdots z_u\}$,$z_u \in L$,其中 L 是标签集,$u \leqslant t$,因此输入输出不对齐。

分类器:给定输入,得到输出。

$$h:(\mathbf{R}^m)^* \mapsto L^*$$

数据集 S:一个由许多 (x,z) 组成的集合,包含了指定的输入和期望的输出。

分类指标:错误率 $\mathrm{LER}(h,S) = \frac{1}{Z}\sum_{(x,z)\in S}\mathrm{ED}(h(x),z)$

Z 是 S 中的标签总数,$\mathrm{ED}(h(x), z)$是编辑距离:通过插入、删除、替换将 $h(x)$变为 z 的最少操作数。

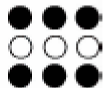
CTC 模型的详细数学推导,见 AlexGraves 的论文:

AlexGraves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks, ICML 2006.

4. 谷歌公司的 CLDNN 模型

从 2012 年开始,深度神经网络在连续语音识别方面也取得了巨大成功。最近一段时间,卷积神经网络和循环网络(LSTM)的实现效果又优于深度神经网络(DNN)。不过这几种神经网络各有特点,所以也各有各的局限之处。

如果把这几种网络有机地组合到一个统一的框架下,性能可以再次得到提升。谷歌



团队针对现有的几种神经网络类型：卷积神经网络、循环神经网络、深度神经网络，进一步提出了一种新的整合模型 CLDNN(CNN+LSTM+DNN)，在语音识别方面获得了新的提升。

CLDNN 是以原始波形为直接输入的网络结构，它主要结合了卷积神经网络(CNN)、循环神经网络(LSTM)和深度神经网络(DNN)。当输入信号进行时间域的卷积操作之后，输出数据再进行一次频率域的卷积操作，以减少频谱的变化，之后再通过三层循环网络，最后再通过一层全连接网络。训练过程中，时间卷积层和其他层会一起进行训练。

CLDNN 的具体组成主要分为 3 部分，如图 2-13 所示。

第一部分是频率建模，采用卷积神经网络，输入数据是以时间为下标的连续向量。

第一部分分为两层：

第一层是 9×9 的频率卷积核(Filter)。

第二层是 3×3 的卷积核。

分别进行卷积，以减少频率的变化种类，采用非重叠(Non-overlapping)的最大池化(Max-pooling)技术。

第二部分：二层循环网络(LSTM)，采用截断的 BPTT(Back-Propagation Through Time)算法，展开 20 个时间步。为了保证连续识别过程中只分析过去的信息，不识别未来的信息，输入中 $r=0$。

第三部分：二层深度神经网络，很容易分别出高阶特征，从而将数据分类。

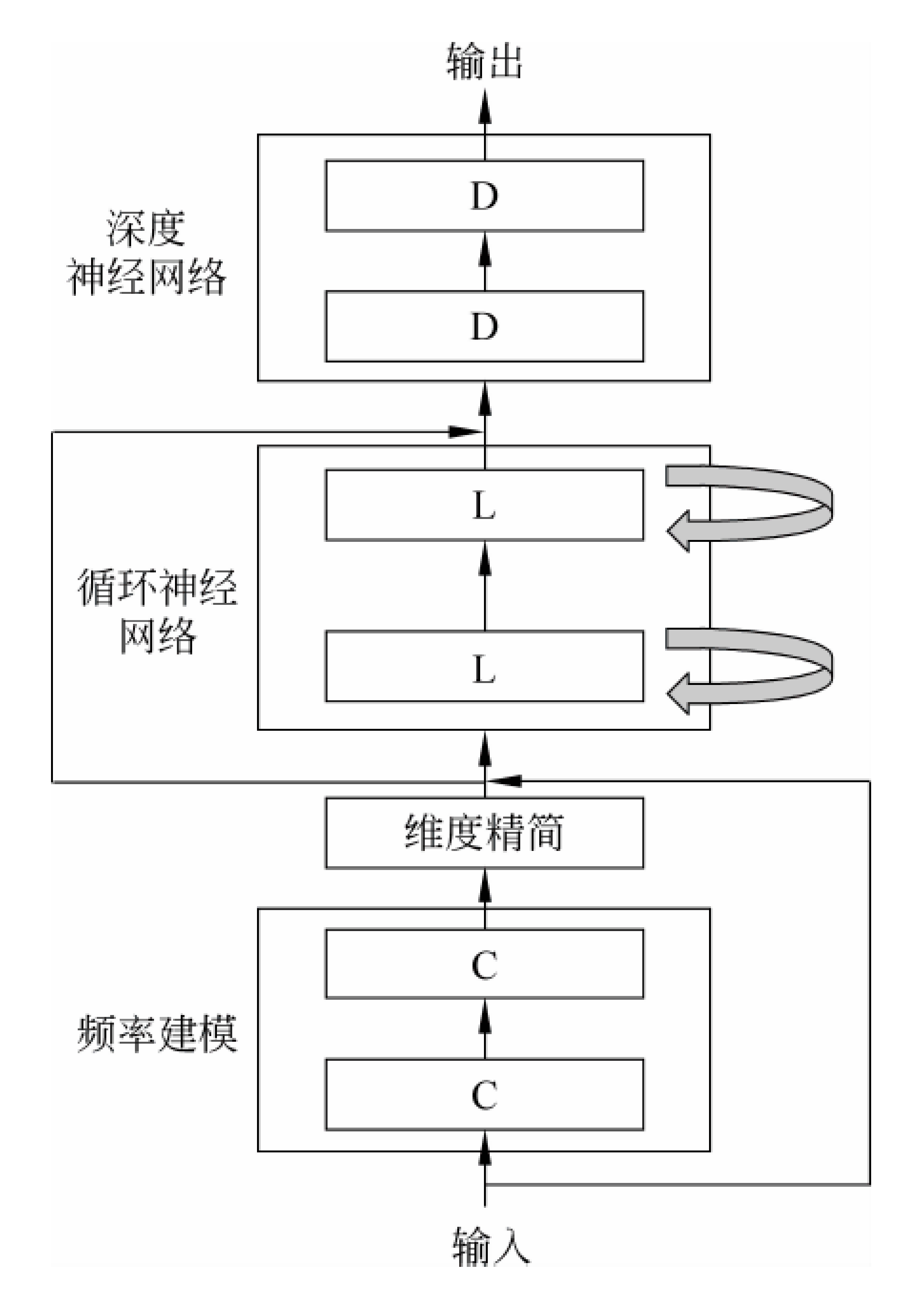


图 2-13 CLDNN 的结构图

LSTM 对语料进行单帧(某特定时刻的音频信息)特征分析，但是分析获得的特征是有限的。而一些未分析出的高阶信息可以帮助人们更好地识别上下文关系。CNN 可以分析出这些高阶特征，所以可以将两者组合。

相比其他网络结构，CLDNN 的两处改进如下。

(1) 将输入中的短时特征传输给 LSTM 层。

(2) 将 CNN 的输出传输给 LSTM 和 DNN,以获得更多的特征数据。

在 LSTM 中,从输入到输出的映射网络不深,意味着没有中间隐藏的非线性层。若中间变量数减少,则模型会更加高效,输出会更容易预测。这可以通过 LSTM 层后加 DNN 实现。

实验中采用了 3 组不同的数据作为测试集。第一种是 200h 的小语料库。第二种是 2000h 的大语料库,不带噪声。第三种是 2000h 的大语料库,人为加入了噪声。测试的结果表明 CLDNN 的健壮性不错,均取得了 4%~5%的性能提升。

5. 纯 DNN 模型

深度学习采用通用的神经网络来替代复杂的、多维度的机器学习方法。这些神经网络经过训练以后可以用来优化可微分的代价函数或损失函数(Loss/Cost Function)。这种方法已经在语音识别上取得巨大的成功,被称为"纯正"的 DNN 方法。

只要拥有了相当多的训练数据和足够的计算资源,就可以构建一个高水准的大词汇量连续语音识别系统。

1) 百度公司的 DeepSpeech2

百度公司的 DeepSpeech2 的方法是比较典型的语音识别的纯深度神经网络结构。DeepSpeech2 采用 5~9 层的网络,其中隐含层有 1~7 层。网络架构包括一到多个卷积层网络,多个(单向或双向)循环网络,一个全连接网络(FCN),一个 softmax 层网络。

DeepSpeech2 的特点如下。

(1) 输入为频谱图;英文输出为{a, b, c, …, z, space, apostrophe, blank},中文输出为{罗马字母表,6000 个汉字}。

(2) 采用 BatchNorm(Batch Normalization) 方法,加速收敛。

(3) 采用 SortaGrid 方法,保证 CTC 的平稳性(短句优先训练)。

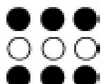
(4) 采用 GRU。GRU 和 LSTM 的准确性相差不大,但 GRU 运算更快。

(5) 采用 Lookahead Convolution 和单向模型,因为双向 LSTM 的时延达不到要求。

2) Nervana 公司的实现

下面介绍一下 Nervana 公司(已被 Intel 公司收购)提出的端到端语音识别模型,采用的是百度 DeepSpeech2 的设计。该端到端语音识别系统由 3 个主要部分组成(见图 2-14)。

(1) 特征提取。将原始音频信号(例如来自 wav 文件)作为输入,并产生特征向量序



列，其中有一个给定音频输入帧的特征向量。

特征提取级的输出示例包括原始波形、频谱图和同样流行的梅尔频率倒频谱系数(Mel-Frequency Cepstral Coefficients，MFCCs)的切片。

(2) 将特征向量序列作为输入，并产生以特征向量输入为条件的字符或音频序列的概率声学模型。

(3) 采用两个输入(声学模型的输出和语言模型)的解码器，在声学模型生成序列的情况下，搜索最可能的转录。

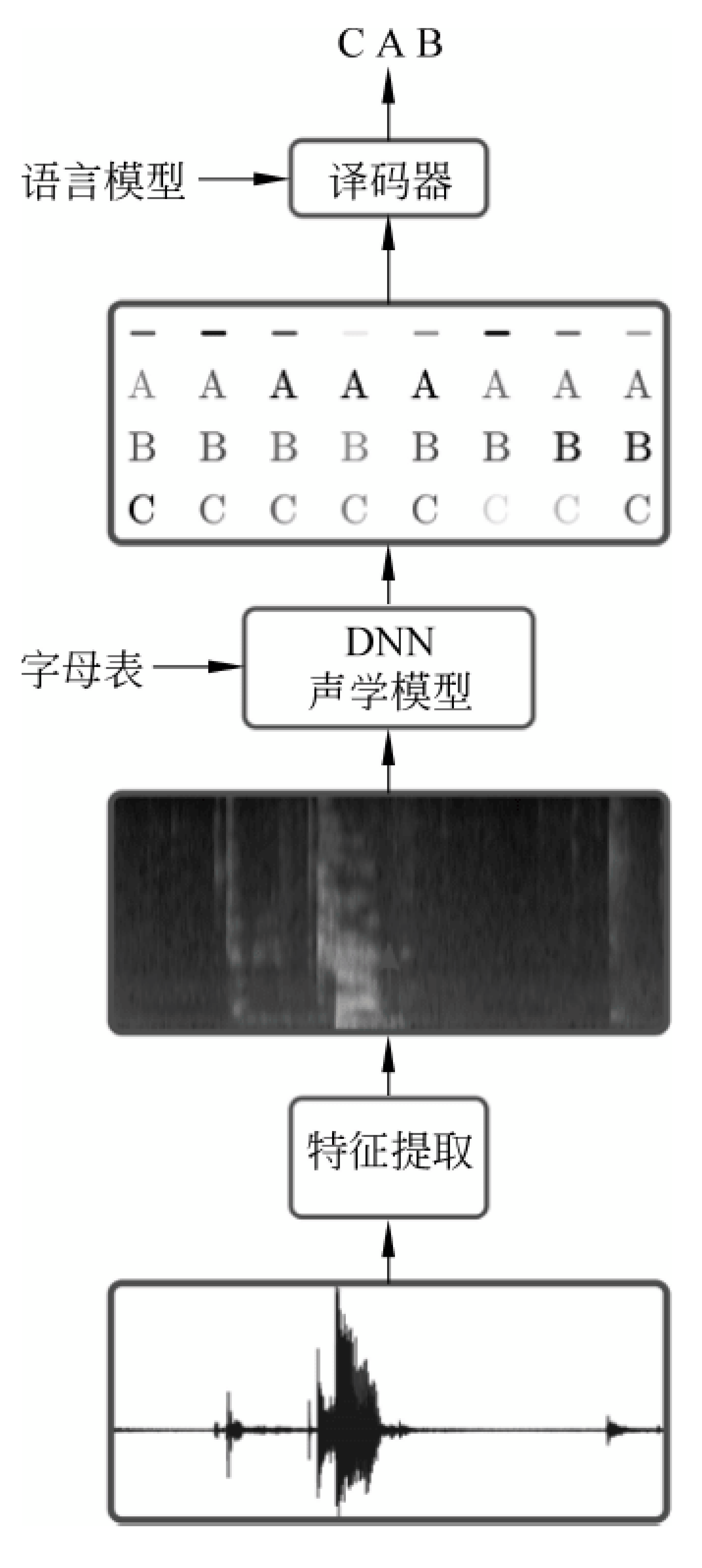


图 2-14　端到端英文语音识别系统

纯 DNN 模型的语音识别，可以参考百度公司的 DeepSpeech 和 DeepSpeech2 论文。

一个具体实现和源代码，可参考如下网址。

(1) https://www.nervanasys.com/end-end-speech-recognition-neon/。

(2) https://github.com/NervanaSystems/deepspeech。

目前各大互联网公司，如百度、科大讯飞、搜狗等，纷纷发布了语音输入法、方言识别、语音转录(如百度公司的 swiftscribe)等功能，也进一步表明端到端深度学习的语音识别的成熟性，以及深度学习技术在语音识别中的日益普及。

2.3.2　计算机视觉

计算机视觉(Computer Vision，CV)包含对象检测、人脸识别、文字识别等。对象检测(Object Detection)是计算机视觉的一项基本功能。下面主要研究一下对象检测技术。

1. R-CNN 系列

AlexNet 在图像识别任务上取得的巨大成功，让目标检测领域的研究者们也将更多的精力投入到基于深度学习的检测方法上。

Ross Girshick 等人提出的 R-CNN(Region-based Convolutional Neural Networks)算法第一次将这个想法变为现实，并在 Pascal VOC 目标检测比赛中取得巨大突破。

R-CNN 的工作流程是首先利用选择性搜索(Selective Search)算法提取候选区域(Proposal),接着通过卷积神经网络对候选区域提取特征,最后通过 SVM 分类器对每个候选区域进行分类识别,将非极大值抑制(Non-Maximum Suppression,NMS)去重后的最终结果返回。

随后,Ross Girshick 和其他研究者对 R-CNN 算法进行了多次改进,包括借鉴 SPP-Net 思想减少候选区域卷积特征重复提取的 Fast R-CNN 算法,以及使用 RPN(Region Proposal Network)代替选择性搜索算法加速候选区域提取的 Faster R-CNN 算法。

2. YOLO 算法

YOLO(You Only Look Once)算法是第一个达到实时性要求的深度学习检测算法,在 Titan X GPU 上的运行速度可以达到 45 帧/秒(Frame Per Second)。和 R-CNN 系列算法事先提取候选区域再对其进行分类识别的思路不同,YOLO 直接将整张图像输入到神经网络中,将目标检测任务化简成一个回归问题,在保证精度不过多损失的前提下,极大地提高了识别速度,实现端到端(End-to-End)的快速目标检测。

YOLOv2 算法在 YOLO 算法的基础上,该研究的作者同样采取了 Anchor 机制,并新加入了 K 均值聚类、批次规范化、转移层、直接位置预测等技巧,使得 YOLOv2 在网络训练时的收敛速度,以及最终检测时的结果准确度和计算速度等方面得到显著提升。

2.4 深度学习实践

2.4.1 建模工具

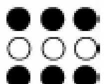
深度学习网络内部建模过程如图 2-15 所示。

深度学习工具是一套灵活的建模框架。使用深度学习工具,在大数据集与大运算量下,采用一些基本框架,避免很多人工干预和手动编程,实现过程的自动化。

一个神经网络模型主要包括数据输入、神经层处理、损失函数计算、反向梯度计算、训练与验证。神经层处理包括网络结构、连接关系,以及激励函数的选择。

2.4.2 软硬件工具

各家互联网巨头都推出了相应的软件工具,主要有谷歌公司的 TensorFlow、脸书公



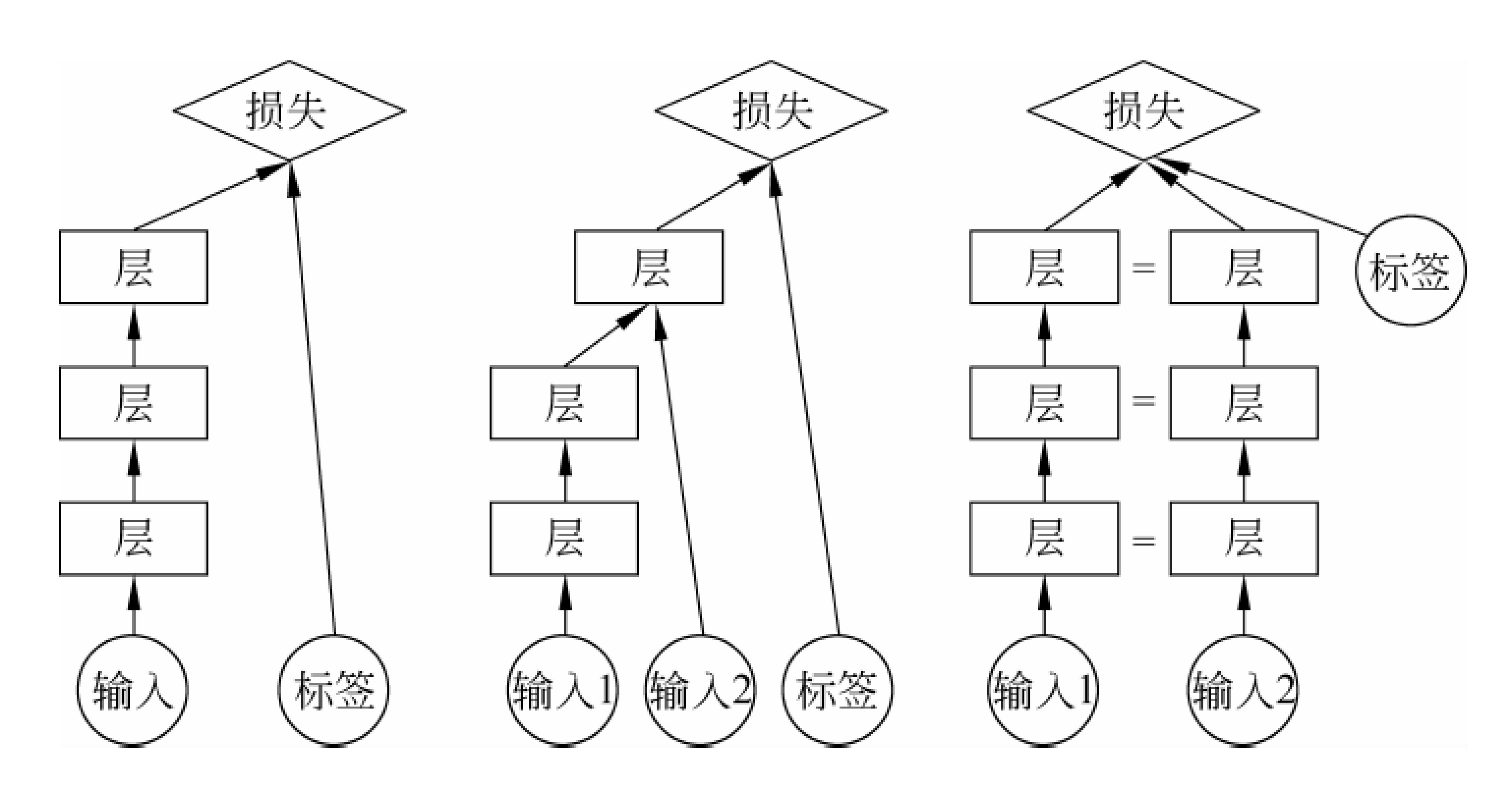


图 2-15　深度学习网络内部建模过程

司的 Torch、微软公司的 CNTK，以及亚马逊公司支持的 MXNET。这些框架将以上的模块和过程自动化，大大促进了深度学习技术的普及。

深度学习软硬件工具如图 2-16 所示。

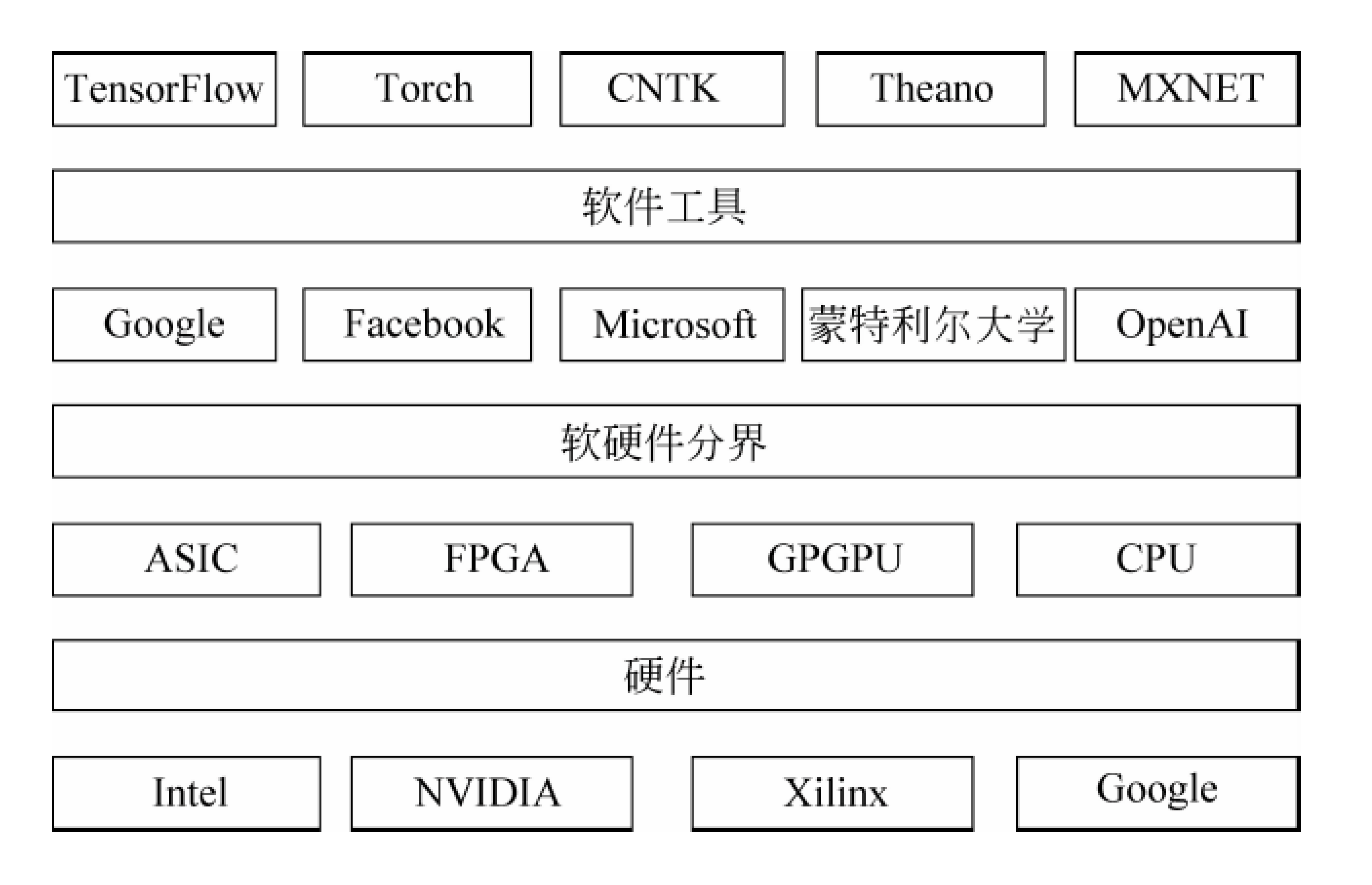


图 2-16　深度学习软硬件工具

深度学习工具一直是深度学习研究进展的一个副产品。国际上对深度学习的研究，呈现出群雄逐鹿的局面，其中当前几个实力雄厚的团队如下。

(1) 由 Jeff Dean 领导的 Googel Brain 团队，包括深度学习领军人物 Geoffrey Hinton。

(2) 由 Yann LeCun 领导的 Facebook AI Research。

(3) 由 Ilya Sutskever 领导的 OpenAI。

从硬件层面，底层硬件包括可编程逻辑阵列（FPGA）、通用图形处理器（GPGPU）、通用处理器（CPU）群集。因为运行效率的问题，目前主要采用通用图形处理器、FPGA 和定制化的方法。

例如，NVIDIA 公司提出的 Pascal 系列、Tesla 系列等通用图形处理器。英特尔公司收购了 Nervana 和 Altera 公司，推出的 Phi 处理器等。谷歌公司更是推出了定制化 ASIC 的 TPU（TensorFlow Processing Unit）。

微软、亚马逊和谷歌都推出了带通用图形处理器的云主机，大大方便了深度学习的使用。几家公司又推出了带可编程逻辑阵列（FPGA）的云主机，大大方便了定制化的神经网络训练和部署。

2.5 小结

本章全面回顾了深度学习的基本原理和技术。首先，介绍了人工神经元的结构，多层人工神经网络的训练过程。接着，介绍了人工神经网络的典型架构，如卷积神经网络、循环神经网络和长短时记忆网络。最后，介绍了机器感知任务中的语音识别和计算机视觉的技术突破和最新进展。

当前深度学习的主要软硬件技术框架日益成熟，尤其在数据中心中已经大规模部署。各大互联网企业都投入巨大人力和物力，纷纷开发了各自的软件框架，并配合各自的硬件平台，完善各自的机器学习应用的生态环境。

参考文献

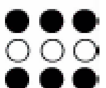
[1] LeCun Y, Bengio Y, Hinton G. Deep learning[J]. Nature, 2015, 521(7553):436-444.

[2] Le Quoc V. A Tutorial on Deep Learning Part 1: Nonlinear Classifiers and The Back-propagation Algorithm [EB/OL]. 2015. https://cs. stanford. edu/ ~ quocle/tutorial1. pdf.

[3] Le Quoc V. A Tutorial on Deep Learning Part 2: Autoencoders, Convolutional Neural Networks and Recurrent Neural Networks[EB/OL]. 2015. https://cs. stanford. edu/ ~ quocle/tutorial2. pdf.

[4] denoising autoencoder[EB/OL]. http://deeplearning. net/tutorial/dA. html.

[5] Bengio Y, Lamblin P, Popovici D, et al. Greedy Layer-wise Training of Deep Networks[C]// International Conference on Neural Information Processing Systems. Cambridge: The MIT Press,



2006:153-160.

[6] Rumelhart D E, Hinton G E, Williams R J. Learning Internal Representation by Back-propagation of Errors[J]. Nature, 1986, 323(323):533-536.

[7] Dean J. Large-Scale Deep Learning for Building Intelligent Computer Systems[C]// ACM International Conference on Web Search and Data Mining. ACM, 2016:1-1.

[8] Chan W, Jaitly N, Le Quoc V, et al. Listen, Attend and Spell[J]. Computer Science, 2015.

[9] Graves A, Mohamed A, Hinton G. Speech Recognition with Deep Recurrent Neural Networks[C]. ICASSP 2013, 38(2003):6645-6649.

[10] Zhang S, Liu C, Jiang H, et al. Feedforward Sequential Memory Networks: A New Structure to Learn Long-term Dependency[J]. Computer Science, 2015.

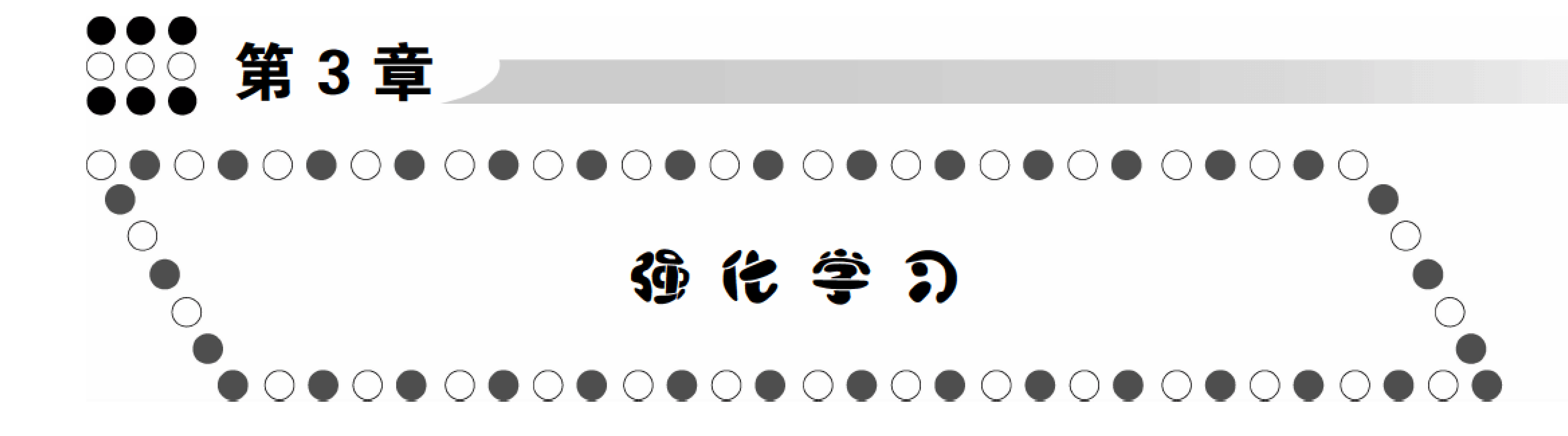
[11] 李航. 统计学习方法[M]. 北京:清华大学出版社, 2012.

[12] 同济大学数学教研组. 高等数学(上)(下)[M]. 北京:高等教育出版社,1996.

[13] 同济大学应用数学系. 线性代数[M]. 4版. 北京:高等教育出版社, 2003.

[14] Norman P Jouppi, et al. In-Datacenter Performance Analysis of a Tensor Processing Unit[C]. 44th International Symposium on Computer Architecture (ISCA), Toronto, Canada, June 26, 2017.

[15] Uijlings J, Sande K, Gevers T, et al. Selective Search for Object Recognition[J]. International Journal of Computer Vision, 2013, 104(2):154-171.



第 3 章 强化学习

强化学习是机器学习中的一个领域，强化学习强调如何基于环境而行动，以取得最大化的预期利益。它是一种通用的学习与规划(Learning and Planning)的方法框架。

3.1 强化学习基础

3.1.1 强化学习概述

1. 强化学习的原理

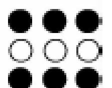
强化学习(Reinforcement Learning)是一种通用的决策框架(Decision-Making Framework)。在强化学习中，智能体(Agent)具有采取动作(Action)的能力(Capacity)，每次动作都会影响智能体的未来状态(State)，返回一个标量的奖赏信号(Reward Signal)来量化表示成功与否(Success)。强化学习算法的目标(Goal)就是如何采取动作(Action)最大化未来的奖赏(Future Reward)。强化学习示意图如图 3-1 所示。

简单地说，强化学习是使用反馈来更新行动，以获得最大化收益的算法。而深度学习是表示学习的通用框架。给定目标，学习到需要达到目标的表示。深度学习使用最小的领域知识，特征表示直接从原始输入中自动学习获得。

2. 强化学习的观测

在每个时间步 t，智能体的行为如下。

(1) 执行动作 a_t。



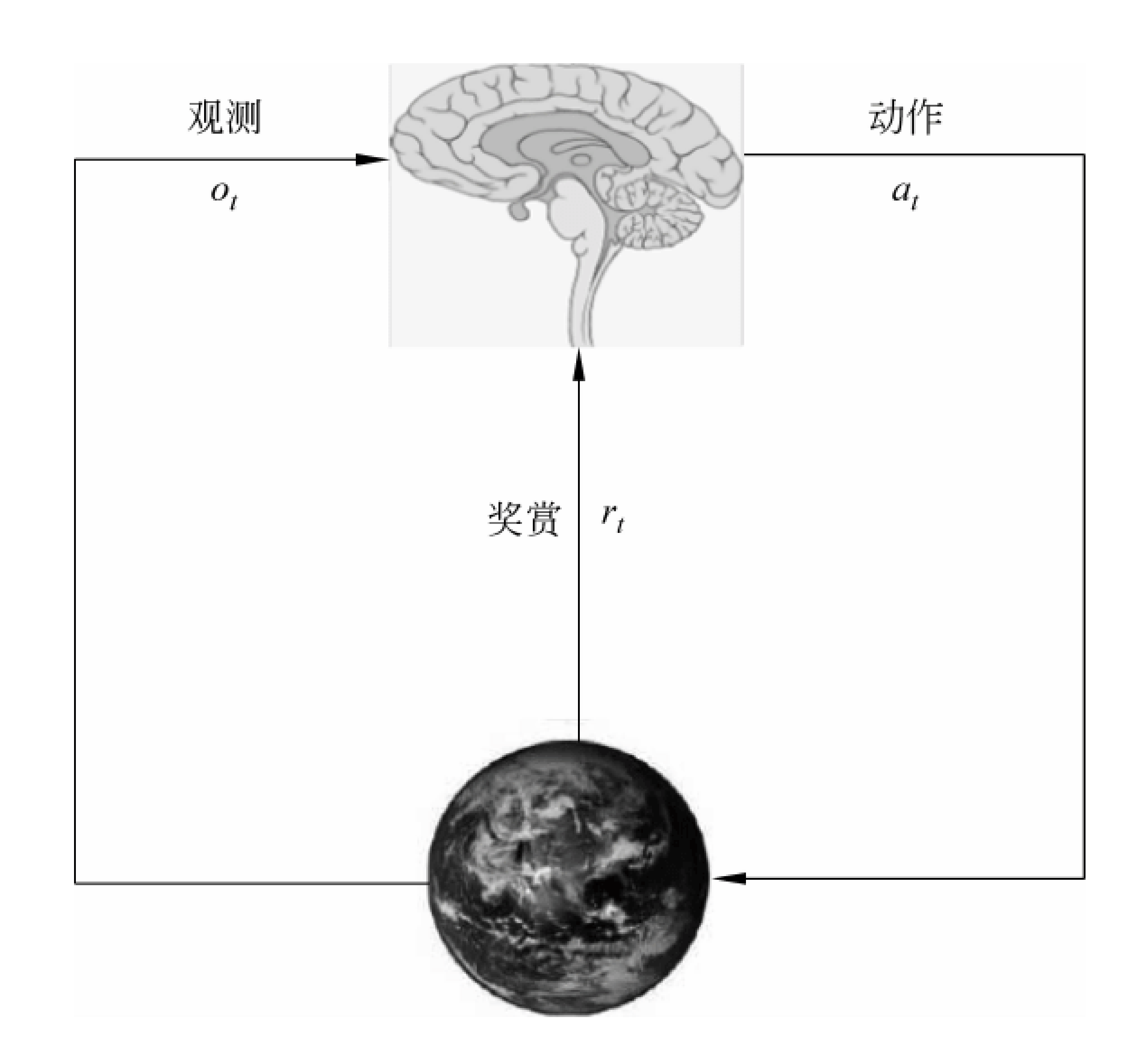


图 3-1　强化学习示意图

（2）获得一个环境的观测 o_t。

（3）获得一个标量的奖赏 r_t。

外界环境的行为如下。

（1）接收到智能体的动作 a_t。

（2）发出一个观测 o_{t+1} 给智能体。

（3）发出一个奖赏 r_{t+1}。

这里经验(Experience)就是一组观测、动作、奖赏的序列。

$$o_1, r_1, a_1, \cdots, a_{t-1}, o_t, r_t$$

状态 s_t 可以认为是经验的函数，即

$$s_t = f(o_1, r_1, a_1, \cdots, a_{t-1}, o_t, r_t)$$

在一个完全可观测的环境，以上公式简化为

$$s_t = f(o_t)$$

3. 强化学习的要素

从强化学习智能体的角度看，强化学习包含一组要素组件。

（1）策略(Policy)是指智能体的行为函数。

（2）价值(Value)函数是指每个状态与动作的成效。

(3) 模型(Model)是指智能体对环境的建模表示。

按照要素的区分,强化学习的方法包括基于价值的强化学习(Value-based RL)、基于策略的强化学习(Policy-based RL)和基于模型的强化学习(Model-based RL)。

基于价值的强化学习在于在任何策略下,估计最优的价值函数。

基于策略的强化学习在于寻找最优策略 π^* 。

策略是智能体的行为,将状态映射为动作,例如,有如下确定性策略和随机策略。

(1) 确定性策略(Deterministic Policy):$a=\pi(s)$。

(2) 随机策略(Stochastic Policy):$\pi(a|s)=P[a|s]$。

3.1.2 深度强化学习

深度强化学习(Deep Reinforcement Learning, Deep RL)就是把强化学习 RL 和深度学习 DL 结合起来。用强化学习定义目标,用深度学习给出相应的机制,如 Q 学习等技术,以实现通用人工智能。深度强化学习的核心原理是使用深度神经网络来近似表示强化学习中的价值函数、策略和模型。

深度强化学习又分为基于价值的深度强化学习(Value-based Deep RL)、基于策略的深度强化学习(Policy-based Deep RL)和基于模型的深度强化学习(Model-based Deep RL)。

1. 基于价值的深度强化学习

1) 深度 Q 网络

价值函数(Value Function)就是预测未来的奖励。Q 值函数可以给出长期的期望收益,在策略 π 的情况下,一般使用折扣因子。

$$Q^{\pi}(s,a)=E[r_{t+1}+\gamma r_{t+2}+\gamma^2 r_{t+3}+\cdots \mid s,a]$$

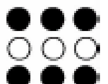
模型是从经验中学习的,可以作为环境的一个代理。可以使用预先查找算法来模拟。

价值函数可以用使用权重为 w 的 Q 网络(Q-network)来表示,如图 3-2 所示,即,

$$Q(s,a,w)=Q^*(s,a)$$

2) Q 学习

Q 学习(Q-learning)就是优化 Q 值应遵循 Bellman 方程,写成如下 Bellman 方程形式:



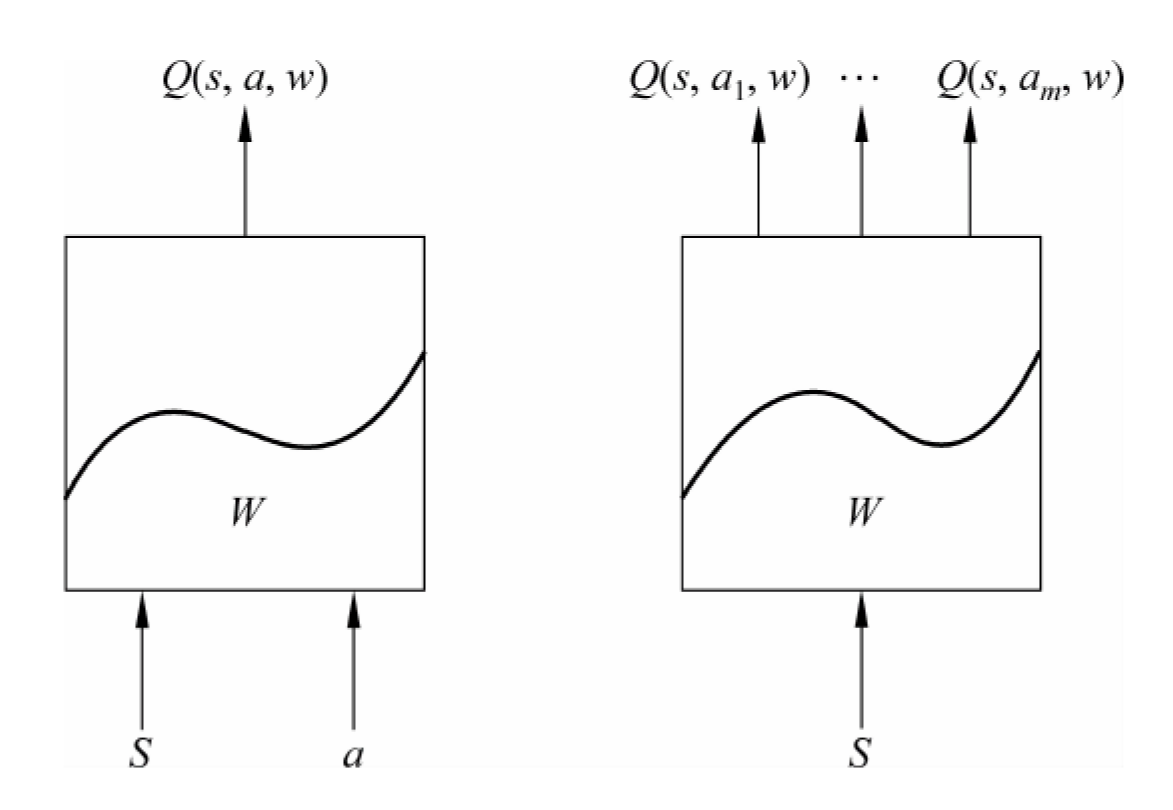


图 3-2　价值函数用权重为 w 的 Q 网络来表示

$$Q^{*}(s,a)=E_{s'}[r+\gamma \max_{a'} Q(s',a')^{*} \mid s,a]$$

Q 学习把公式中的右侧视为一个整体，人们可以使用深度神经网络去近似 Q 值函数，采用梯度下降法最小化损失函数的均方误差的学习过程。损失函数形式表示如下：

$$l=(r+\gamma \max_{a} Q(s',a',w)-Q(s,a,w))^{2}$$

通过这样训练生成的深度神经网络，又称为深度 Q 网络(Deep Q-Network)。

3）经验回放

Q^{*} 收敛可以使用表查询表示方法，但是使用深度神经网络的方法会出现分歧问题(diverge)。这是样本之间的相关性和目标的非平稳性造成的。

为了解决这个问题，消除学习样本的相关性，采用经验回放(Experience Replay)的方法，如图 3-3 所示。

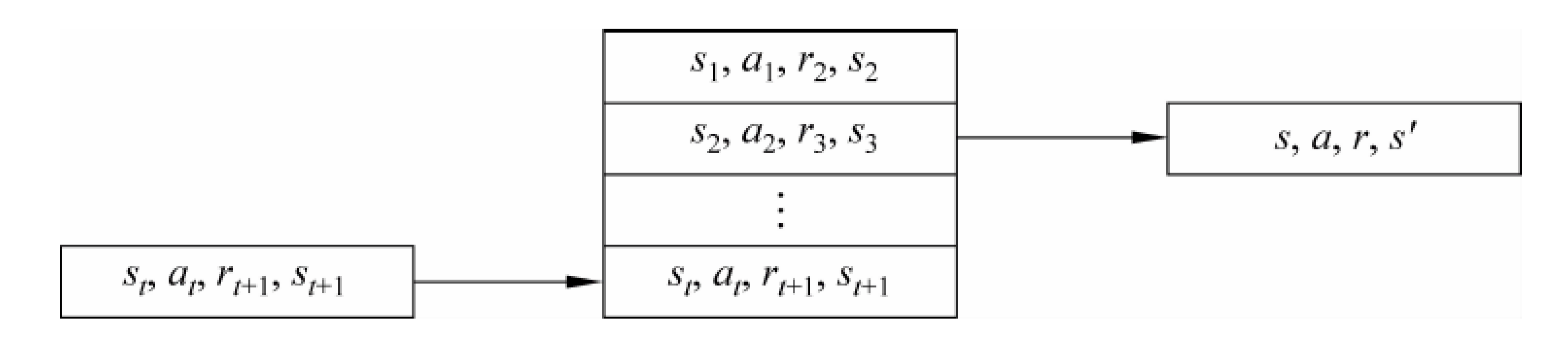


图 3-3　经验回放

从数据集中采样经验，并在损失函数中更新。为了解决非平稳性问题，目标的参数 w^{-} 是固定的。

$$l=(r+\gamma \max_{a} Q(s',a',w^{-})-Q(s,a,w))^{2}$$

2. 基于策略的深度强化学习

1）深度策略网络（Deep Policy Network）

通过深度神经网络（权重参数 u）表示策略：

$$a = \pi(a \mid s, u) \quad 或 \quad a = \pi(s \mid u)$$

定义目标函数是整体折扣后的回报，一般使用折扣因子 γ。

$$L(u) = E[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots \mid \pi(s, u)]$$

通过随机梯度下降法来优化目标函数，例如修正权重 u 来获得更多的回报。

一个随机策略 $\pi(a|s, u)$ 的梯度为

$$\frac{\partial l(u)}{\partial u} = E\left[\left(\frac{\partial(\ln\pi(a \mid s,u))}{\partial u} Q^{\pi}(s,a)\right)\right.$$

如果 a 是连续可微分的，一个确定性策略 $a = \pi(s|u)$ 的梯度可以进一步简化为

$$\frac{\partial l(u)}{\partial u} = E\left[\left(\frac{\partial(Q^{\pi}(s,a))}{\partial a} \frac{\partial a}{\partial u}\right)\right.$$

注：全书中出现的对数没有底数，是因为所参考的资料中不限制 log 的底数，默认是自然对数底。

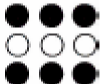
2）Actor-Critic 算法

过去的时间里，人们提出了单表演者（Actor-only）、单评论家（Critic-only）、表演者-评论家（Actor-Critic）等强化学习方法。其中，表演者（Actor）就是策略（Policy）函数的同义词，评论家（Critic）就是价值（Value）函数的同义词。

Actor-only 方法一般采用一组参数化的策略来工作，这样可以直接使用优化过程。这种参数化策略的优点是连续动作的范围可以生成，但是优化方法（如策略梯度方法）会在梯度估计时出现方差高的问题，这又将导致学习比较慢的问题。

Critic-only 方法使用时间差分学习，具有期望回报估计方差低的优点。从 Critic-only 方法直接导出策略的方法是选择贪婪动作。选择那些价值函数指示是平均回报最高的动作。然而，这需要诉诸于优化的流程，在进入每个状态时，找到最大价值的动作。当动作空间是连续时，这是一个计算密集的任务。所以，Critic-only 方法一般会离散化连续动作空间，此后，在动作空间的优化就变成一个枚举过程。

Actor-Critic 方法组合 Actor-only 和 Critic-only 方法的优点。尽管参数化的 Actor 可以计算连续动作，无须在价值函数上优化，而 Critic 的优点是为 Actor 提供关于行为效果的低方差的知识。更具体地说，Critic 对平均回报的估计允许 Actor 用来更新梯度，使



其具有的方差较小，这就加速了学习过程。Actor-Critic 方法相比于 Critic-only 方法，具有更好的收敛属性。所以，Actor-Critic 方法的良好属性，使其成为在实际应用中首选的增强学习算法。

Actor-Critic 算法的估计价值函数为 $Q(s,a,w) \approx Q^{\pi}(s, a)$。通过随机梯度下降法更新权重。

$$\frac{\partial l}{\partial u} = E\left[\left(\frac{\partial(\ln\pi(a \mid s,u))}{\partial u}Q(s,a,w)\right)\right.$$

如果 a 是连续可微分的，采用确定性策略 $a = \pi(s|u)$ 的梯度可以进一步简化为

$$\frac{\partial l}{\partial u} = E\left[\left(\frac{\partial(Q(s,a,w))}{\partial a}\frac{\partial a}{\partial u}\right)\right.$$

3. 基于模型的深度强化学习

基于模型的深度强化学习(Model-based Deep RL)试图对外界环境建模，创建预测模型(Predictive Model)，从而预测反馈的奖赏和观察。对于一些计算机游戏，如 Atari，可以通过卷积神经网络和循环神经网络对输入的人类选手数据进行训练，建立预测模型。但是对于计算机围棋等应用，因为一般无法直接建立模型，所以多采用模型无关的深度强化学习(Model-free Deep RL)方法。

4. 深度强化学习的应用

深度强化学习的应用广泛，包括计算机游戏、计算机围棋、计算机扑克、机器人运动控制。基于价值的强化学习，来训练神经网络 DQN，可学会玩各类游戏，如 Atari 等。

2015 年 11 月，谷歌的 DeepMind 计算机围棋 AlphaGo，首次战胜人类专业棋手。2017 年 8 月，OpenAI 的计算机游戏智能体在 *Data*2 游戏中，首次击败了人类最好选手。

3.1.3 强化学习框架

2016 年 11 月，Google DeepMind 公司开源了 Lab 软件，Lab 源代码托管在 Github 上，见 https://github.com/deepmind/lab。

2016 年 12 月，OpenAI 组织也开源了 OpenAI Universe 软件，Universe 源代码托管在 Github 上，见 https://github.com/openai/universe。

3.2 计算机围棋

3.2.1 围棋游戏

1. 围棋规则

围棋是一项历史悠久的棋类项目。围棋棋盘是 19×19 的网格，纵横各 19 路直线，形成 361 个交叉点。围棋棋子分为黑白两种，白子 180 颗，黑子 181 颗，共 361 颗棋子。

围棋规则比较简单，主要有以下 7 条规则。

(1) 博弈黑白双方，依次落子。落子在纵横网格的交叉点上。

(2) 黑方先走，落黑子；白方后走，落白子。

(3) 在轮到一方落子时，一方可以选择放弃落子。如果双方都放弃，比赛结束。

(4) 下子后不能造成和以前相同的棋局。

(5) 一片同色棋子的所有连线相邻的交叉点(纵横线)，都被对手占满后，则这一片同色棋子就被对手吃掉而清空。

(6) 一方的领土是他的棋子占的位置以及包围的交叉点。

(7) 占位多的一方获胜。

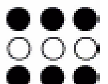
围棋的水平可分为初学者、业余与职业三类。其中初学者(Beginner)分为 30 级，业余(Master)分为 7 段，职业(Professional)分为 9 段，职业 9 段是顶级水准。成为职业 9 段的选手，不仅需要有天赋，还需要付出数十年的时间和精力。职业 9 段的棋力水准也是计算机围棋需要达到的目标。围棋的段位分级如图 3-4 所示。



图 3-4　围棋的段位分级

2. 计算机围棋

围棋是一项搜索空间巨大的棋类游戏，被称为人工智能挑战的一项王冠。计算机棋类游戏被认为是 AI 界的果蝇(Fruit Fly)，是人们研究最多的 AI 游戏。这类游戏提供了



一个验证新想法，与人类较量水平，以及 AI 算法之间容易互相比较的实验沙箱。

2009 年，计算机围棋 Fuego 第一次在 9×9 棋盘上击败 9 段选手。在接下来的几年里，在有让子的情况下，计算机围棋在 19×19 的棋盘上也能击败人类职业选手。

东京电气大学杯(UEC 杯)是比较知名的计算机围棋的比赛。自 2007 年开始，已经累计办了 10 届。比较知名的计算机围棋有 Fuego、CrazyStone、Zen、Erica、Darkforest 等，人们使用这些围棋先后参赛并获得名次。

计算机围棋 CrazyStone，曾经获得第 1 届、第 2 届、第 6 届和第 8 届的 UEC 杯冠军。其作者 Remi Coulom 曾经是法国里尔第三大学的副教授。

计算机围棋 Fuego 获得第 4 届 UEC 杯冠军，其作者 Martin Müller 是加拿大阿尔伯塔大学的计算机系教授。

计算机围棋 Erica 获得第 5 届 UEC 杯亚军，其作者是黄世杰，也是阿尔法围棋的作者之一。计算机围棋 Darkforest 获得第 9 届 UEC 杯亚军，其作者是 Facebook 公司的田渊栋。

计算机围棋 Zen 曾经获得第 5 届、第 7 届和第 9 届的 UEC 杯冠军，其作者是日本的围棋程序员 Yoji Ojima。2016 年，受阿尔法围棋的深度学习算法的启发，其作者将 Zen 升级为 DeepZenGo。

2017 年，腾讯公司的计算机围棋程序绝艺(Fineart)击败 DeepZenGo，获得了第 10 届 UEC 杯冠军。

关于计算机科学家征服计算机围棋的计划，参见以下文献：

Sylvain Gelly, Levente Kocsis, Marc Schoenauer, et al. The Grand Challenge of Computer Go: Monte-Carlo Tree Search and Extensions. Communications of the ACM, Vol. 55, no. 3 (2012): 106-113.

3.2.2 蒙特卡洛树搜索

自从 2006 年的蒙特卡洛树搜索(Monte-Carlo Tree Search, MCTS)算法出现后，计算机围棋的棋力增长有了突破性的进展。下面介绍一下蒙特卡洛树搜索方法。

1. 蒙特卡洛树搜索介绍

蒙特卡洛仿真(Monte-Carlo Evaluations，MCEs)最早由 Abramson 在 1990 年引入，用于解决棋类等游戏。蒙特卡洛树搜索如图 3-5 所示。

Remi Coulom 首先提出在蒙特卡洛树上进行选择(Selectivity)和回溯(Backup)过程。Remi Coulom 开发了商业化围棋软件 CrazyStone。CrazyStone 采用提出的 MCTS 方法,棋力水平大增,曾经是水平最高的计算机围棋程序,棋力水平在业余 5～6 段。

蒙特卡洛搜索树 T 包含节点 $n(s)$,s 代表仿真中的一个状态。每个节点都有一个状态计数值 $N(s)$和动作值 $Q(s,a)$,以及每个动作的计数值 $N(s,a)$,其中 $a\in A$。

蒙特卡洛树搜索包含两部分,即树策略(Tree Policy)和默认策略(Default Policy)。树策略用来选择动作,每次仿真产生状态 $\{s_0,a_0,s_1,a_1,\cdots,s_T\}$,其中$s_T$ 为最终结果状态。每次仿真更新每个状态 s 的计数值 $N(s_t)$和动作值 $Q(s_t,a_t)$。

更新过程如下:

$$N(s_t)\leftarrow N(s_t)+1$$

$$N(s_t,a_t)\leftarrow N(s_t,a_t)+1$$

$$Q(s_t,a_t)\leftarrow Q(s_t,a_t)+\frac{z-Q(s_t,a_t)}{N(s_t,a_t)}$$

最简单的贪婪树策略,就是从 s_0状态出发,选择 Q 值最大的值。

蒙特卡洛树搜索原理如图 3-5 所示。蒙特卡洛树搜索原理的详细步骤如图 3-7～图 3-11 所示。其中图 3-6 给出各图中的符号说明。

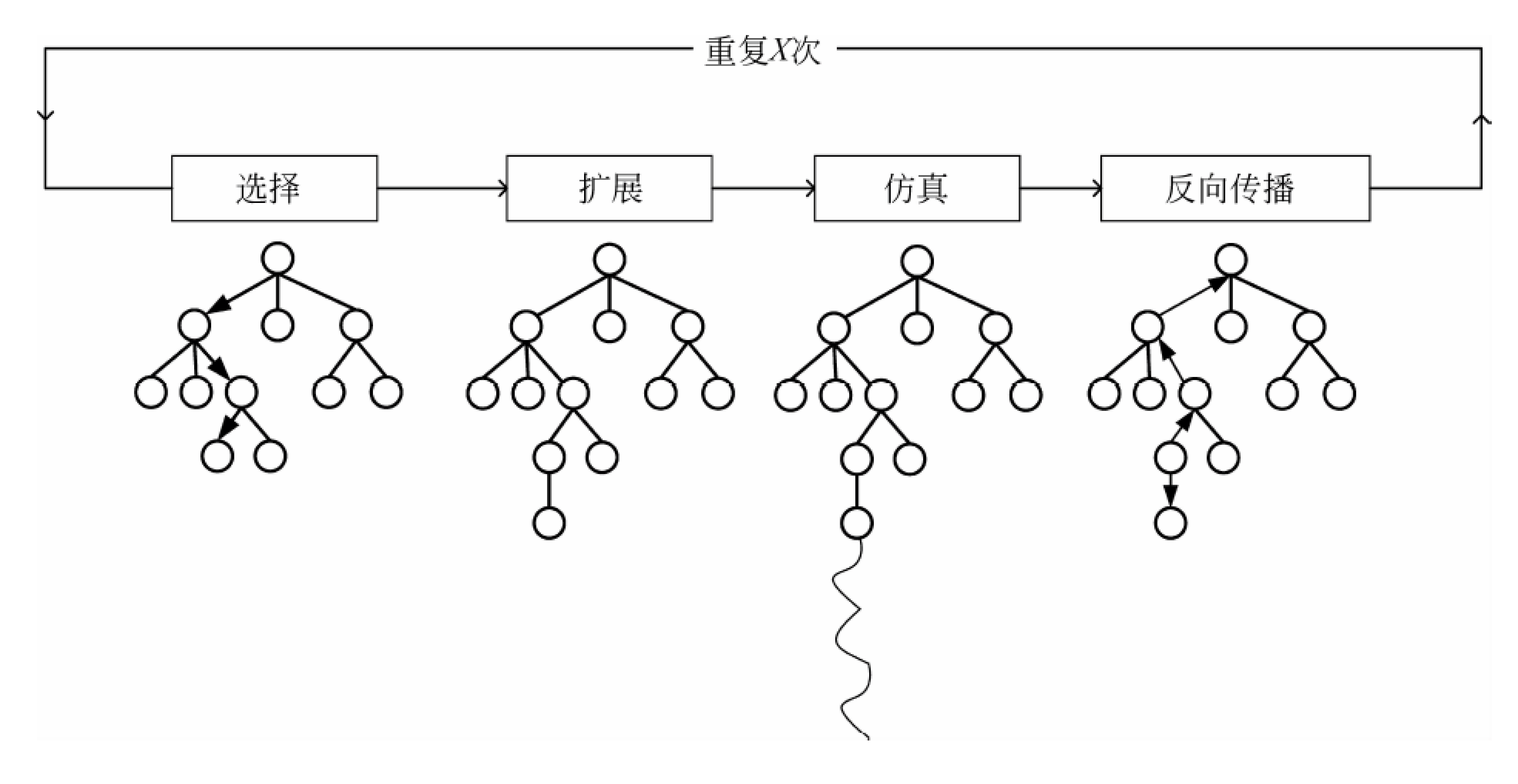
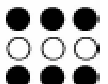


图 3-5 蒙特卡洛树搜索



树的新节点
树中保存的节点
被访问但未保存的节点
比赛结果
当前仿真
先前仿真

图 3-6　符号说明

当前状态
1/1
树策略
默认策略
水平横线的上部分为树策略，下部分为默认策略
1

图 3-7　示例图 1

当前状态
1/2
0/1
树策略
默认策略
0

图 3-8　示例图 2

当前状态
2/3
1/1
0/1
树策略
默认策略
1

图 3-9　示例图 3

当前状态
2/4
1/2
0/1
0/1
树策略
默认策略
0

图 3-10　示例图 4

当前状态
3/5
2/3
0/1
0/1
1/1
树策略
默认策略
1

图 3-11　示例图 5

图中内容取自于 Sylvain Gelly 和 David Silver 的研究论文。第一作者 Sylvain Gelly 也是计算机围棋 MoGo 的设计者，第二作者 David Silver 是阿尔法围棋架构主设计师。该论文提出的思路在阿尔法围棋的核心算法中都有体现，只是更加具体化。可参考的论文信息如下：

Sylvain Gelly，David Silver. Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go. Artificial Intelligence 175，no. 11：1856-1875，2011.

MCTS 方法的效果取决于搜索树的生成。搜索树的每个节点的展开需要一定的策略。如何有效地展开节点，保障生成高质量的节点，又不使得搜索树过度膨胀？

这个问题在数学上被称为多臂赌博机问题(Multi-Armed Bandit，MAB)。UCT (Upper Confidence bounds applied to Trees)算法把树节点的展开生成过程，看作是一个多臂赌博机问题，从而有效地解决了这个问题。下面首先介绍一下多臂赌博机问题和 UCT 方法。

2. 多臂赌博机

多臂赌博机问题：一个赌徒面前有一系列赌博机(或老虎机)，在投入硬币后，每个赌博机返回的回报是不同的。赌徒面临的问题是如何最大化自己的收益。

当赌徒尝试了一系列赌博机后，会获得一些统计上的收益。但是，赌徒并不知道赌博机背后的真实收益分布。在获得已有的收益后，赌徒遇到的策略问题是：是继续专注于当前获得的收益呢，还是去尝试更多的赌博机？

赌徒如果专注于已获得收益的赌博机，至少可以保持一定的收益。如果去尝试更多的先前未测试过的赌博机，有可能出现尝试失败的情况，但也有可能会发现具有更大收益的赌博机。

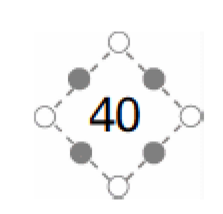
3. UCB 方法和 UCT 方法

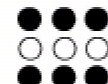
UCB 方法是针对多臂赌博机问题的一种解法。该方法力图在探索(未知的赌博机)和保持(现有的回报)之间找到平衡。

UCB 方法的全称是 Upper Confidence Bounds，即置信上界方法。UCB 算法最早由 Peter Auer 等人提出。

UCT 的全称是 Upper Confidence bounds applied to Trees，它是 UCB 在游戏树(Game Tree)上的一个应用，首先由 Kocsis 和 Szepesvari 在 ECML 论文中提出。

UCT 算法把蒙特卡洛树节点的展开生成过程，看作是一个多臂赌博机问题。通过





将 UCB 方法移植到蒙特卡洛树上，有效地解决了这个问题。

4. UCT 方法与 EE 问题

UCT 方法的核心思想是解决短期回报与长期回报之间的折中问题，即 EE 问题。EE 问题的全称是(Exploitation or Exploration)，或者 Exploit-Explore 问题。其中：

(1) 保持(Exploit)是指对比较确定有回报的方向，继续尝试。

(2) 探索(Explore)是指对新的方向，不断探索，防止一模一样的确定方向。

UCT 方法赋予每个动作一个动作值(Action Value)，表示如下：

$$\widetilde{X}_j(t)+\sqrt{\frac{21nt}{T_{j,t}}}$$

其中，前项是该分支目前的收益均值，后面的项称为探索激励(Bonus)。探索激励本质上是均值的标准差形式，其中 t 是目前的试验次数，$T_{j,t}$是这个分支被试验的次数。

这个公式表明：均值越大，标准差越小，被选中的概率会越来越大，起到保持的作用；同时，那些被选次数较少的赌博机，也会得到被试验机会，起到探索的作用。

阿尔法围棋使用了 PUCT(Predictor + UCT)方法。PUCT 在 UCT 方法上进行了微调。其公式推导如以下论文所示：

C D Rosin. Multi-armed Bandits with Episode Context. Annals of Mathematics and Artificial Intelligence, March 2011, Volume 61, Issue 3, 203-230.

5. MCTS+UCT 算法示例

MCTS+UCT 算法的原理由以下公式所示。

$$Q^{\oplus}(s,a)+c\sqrt{\frac{\ln N(s)}{N(s,a)}}$$

其中，对数为自然对数。

$$a^{*}=\underbrace{\operatorname{argmax}}_{a}(Q^{\oplus}(s,a))$$

树策略使用 UCT 算法，选择下一步动作，目标是最大化动作的置信上界，即树策略选择动作 a^{*}，最大化增强值 $Q^{\oplus}$。

默认策略使用手工规则、领域专家知识、监督学习训练的模式来选择下一步动作。

UCT 方法赋予每个动作一个动作值，这时会产生图 3-12(a)。从根节点有两个选择，即左节点和右节点，胜率分别为 5/7 和 0/2。我们设定的动作值为

$$SCORE(L) = 5/7 + k.\,sqrt(\ ln(10)/7\ )$$

$$SCORE(R) = 0/2 + k.\,sqrt(\ ln(10)/2\ )$$

如果简单按照胜率做选择，应该选择左节点 L，因为其胜率更高。这时会产生图 3-12(b)。但是，如果考虑探索的奖励项，即公式的最右项，则得到总分值更高。这时会产生图 3-12(c)。

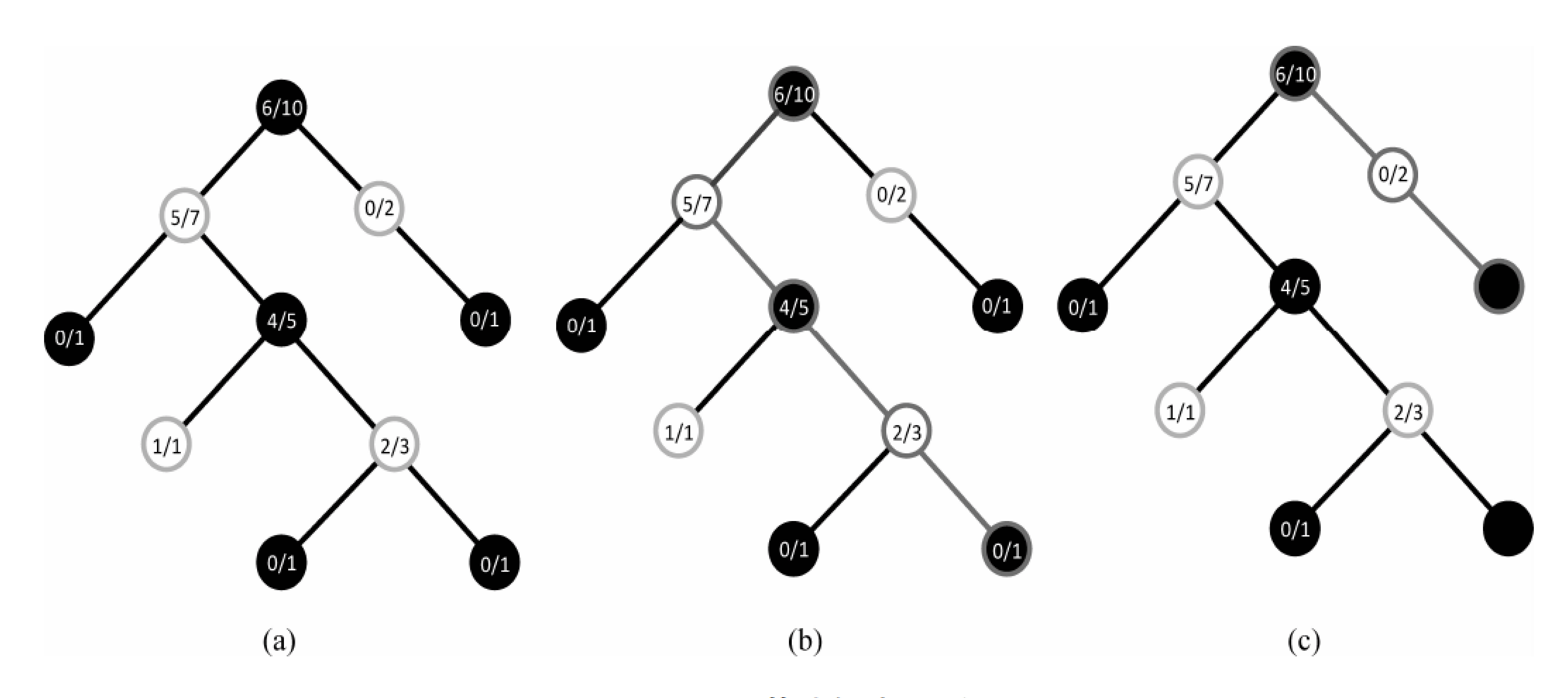


图 3-12　UCT 算法解决 EE 问题

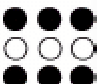
图 3-12 表明，通过调节 k 值，会产生不同的分值。这样，可以使算法同时兼顾胜率(保持)和试验(探索)的次数。

用 UCT 方法生成的结果，在实践中证明比贪婪树方法的效果好很多。MoGo 是第一个用 UCT 方法的计算机围棋程序。UCT 算法把树节点的展开生成过程，看作是一个多臂赌博机问题。

MoGo 最初由 S. Gelly 和 Yizao Wang 开发，团队还包括 Remi Munos 和 Oliver Teytaud。MoGo 在棋力水准上显著超过了其他 9×9 棋盘上的竞争对手，开启了计算机围棋的新时代。

RLGO 是 David Silver 开发的一个计算机围棋程序，它采用由 Markus Enzenberger 和 Martin Muller 的 SmartGo 库。它通过强化学习来训练价值函数来预测从一个给定的状态获得的最终结果，学习过程自然采用游戏的胜负结果作为强化学习的奖赏(Reward)。

具体参见 S. Gelly 和 David Silver 的论文：



S Gelly. A Contribution to Reinforcement Learning: Application to Computer Go. PhD thesis, Universite Paris-Sud, 2008.

David Silver. Reinforcement Learning and Simulation-Based Search in Computer Go. PhD thesis, University of Alberta, 2009.

3.2.3 基于卷积网络的围棋程序

1. NeuroGo

在 1996 年，Markus Enzenberger 就提出了使用神经网络来下围棋的想法，并设计了 NeuroGo 程序。在 2012 年以后，最终在计算机围棋设计中得到广泛应用，其效果显著。

2. DeepGo

DeepGo 是 C. Clark 等人在 ICML 2015 上给出的采用卷积网络下围棋的一种具体实现方法。

阿尔法围棋团队的 C. J. Maddison 等人在 ICLR 2015 上也提出了一种方法。这种方法最终被应用在阿尔法围棋中。

3. Darkforest

Darkforest 是 Facebook 公司的田渊栋博士负责开发的计算机围棋程序，曾获得 UEC 杯的亚军。Darkforest 也是采用卷积网络进行下围棋。Darkforest2 在 KGS 比赛获得 2 段水平，Darkforest3 增加了蒙特卡洛树搜索，在 KGS 比赛达到 5 段水平。

3.3 阿尔法围棋的原理

AlphaGo 是由 Google 的子公司 DeepMind 研发的计算机围棋，又称为阿尔法围棋。阿尔法围棋综合采用蒙特卡洛树搜索、强化学习、卷积网路等技术，第一次在不让子的情况下，击败了人类的职业选手(樊麾，欧洲围棋冠军，当时为职业 2 段)。

阿尔法围棋的胜利，既是算法的胜利，也是深度学习和强化学习的胜利。下面介绍一下阿尔法围棋的研发和工作原理。

3.3.1 阿尔法围棋团队

阿尔法围棋是 DeepMind 的作品。DeepMind 创立于 2011 年，2014 年被 Google 公司收购。

David Silver 是阿尔法围棋的首席架构师。David Silver 也是 UCL（University College London）的 Lecturer（类似讲师），教授“强化学习”课程。

Aja Huang 即 Shih-Chieh Huang（中文名黄世杰），是阿尔法围棋的工程实现技术负责人。计算机围棋 AjaGo 和 Erica 的作者。

David Silver 和 Aja Huang 都在加拿大阿尔伯塔大学做研究，合作导师都是 Martin Muller，所以 David Silver 和 Aja Huang 比较熟。

Chris J. Maddison 是牛津大学统计系博士生，在 DeepMind 做研究科学家（一周两天）。硕士导师是加拿大多伦多大学的 Geoffrey Hinton。

Demis Hassabis 是 Google DeepMind 的创始人。Demis Hassabis 在 UCL 拿到了神经学博士学位，与 David Silver 是校友。

当前的人工智能领域，欧洲和加拿大的学者占有很重要的地位。加拿大的阿尔伯塔大学在计算机游戏和强化学习方面的研究，领先于其他世界一流大学。加拿大多伦多大学和蒙特利尔大学在深度学习方面的研究，领先于其他世界一流大学。

计算机围棋相关人物的具体介绍，可以在如下网站中找到：

https://chessprogramming.wikispaces.com。

3.3.2 深度卷积网络

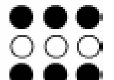
阿尔法围棋使用的 4 个深度神经网络（见图 3-13），均采用 12 层的卷积网络。

这 4 个网络的训练过程和使用具体说明如下。

（1）π_{SL} 是使用监督学习训练的策略网络 $p_\sigma(a|s)$，它是 12 层的卷积网络，σ 表示该卷积网络的内部连接权重。

训练数据：使用 3000 万 KGS（Kiseido Go Server）的 5～6 段选手的对弈局势（Positions）。

训练效果：使用所有的输入特征后，该网络 $p_\sigma(a|s)$ 预测专家下棋招数达到 57% 的准确率。如果仅使用当前棋局和下棋历史信息，预测准确率为 55.7%。



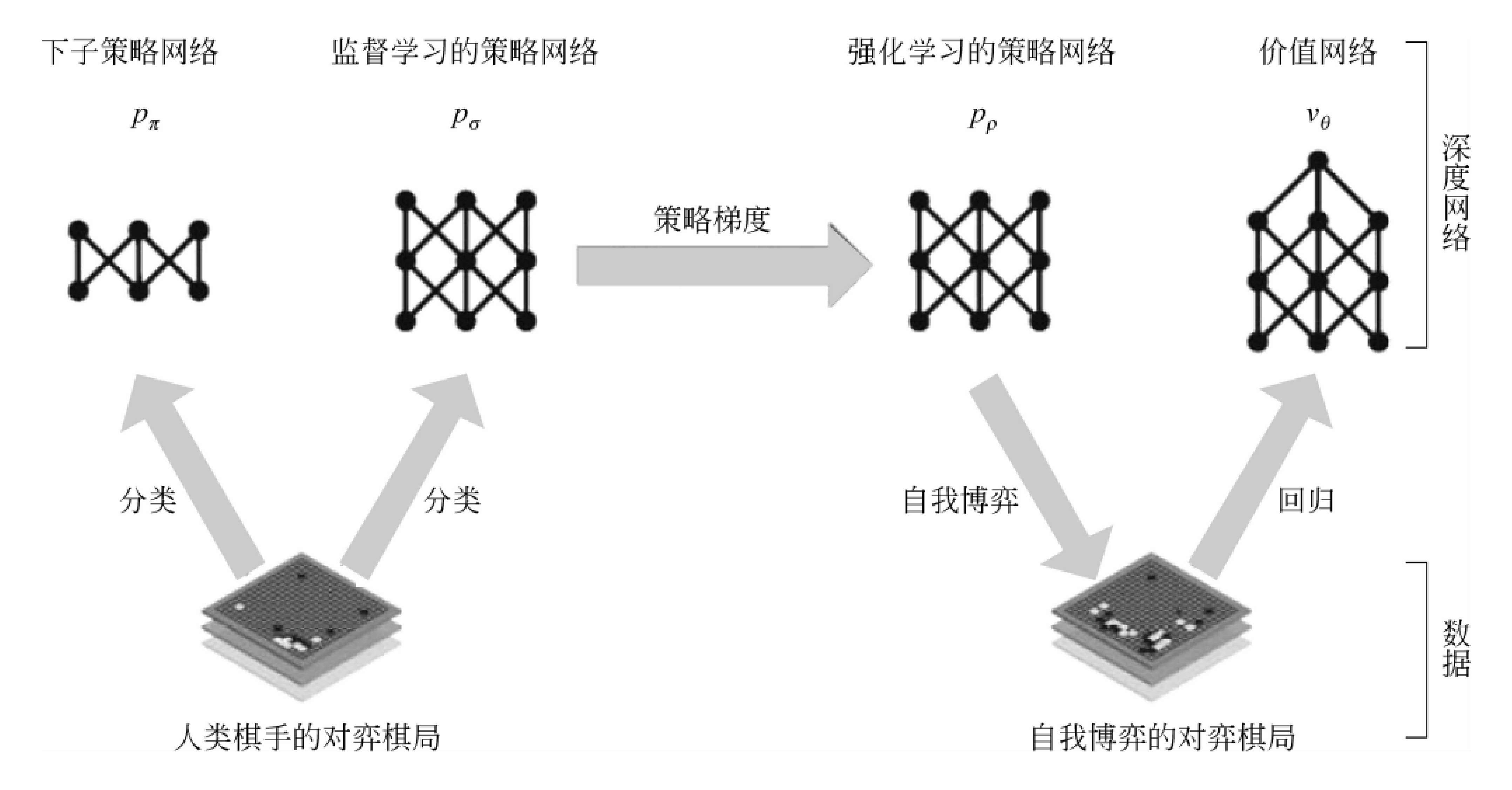


图 3-13 阿尔法围棋使用的 4 个深度神经网络

(2) 策略网络 $p_\pi(a|s)$是一个快速下棋网络。该网络使用部分输入特征，使用 softmax()函数。

策略网络 $p_\pi(a|s)$确定在当前局势(State)下，采取的动作(Action)，用 softmax()函数后，它是一个概率分布。π 表示该卷积网络的内部连接权重。

训练数据：使用 3000 万 KGS 的 5～6 段选手的对弈局势(Positions)。

训练效果：速度快 2μs，但准确率低，仅为 24.2%。

(3) π_{RL}使用增强学习训练的策略网络 $p_\rho(a|s)$。

训练数据：基于监督学习训练的 $p_\sigma(a|s)$，采用策略梯度下降方法，训练 p_ρ，ρ 表示该卷积网络的内部连接权重。

将 ρ 初始化为 σ，即 $\rho=\sigma$。然后将当前版本 ρ^* 与其众多迭代的前版本 ρ'进行自我博弈，将自我博弈产生的棋局数据用来进一步训练。

训练效果：强化学习训练的 $p_\rho(a|s)$对监督学习训练的 $p_\sigma(a|s)$，胜率可达 80%以上。而在无搜索的情况下，该网络对于另一个开源围棋程序 Pachi，胜率可达 85%。

(4) 价值网络 $v_\theta(s)$训练。

训练目标：最小化预测值 $v_\theta(s)$与最终结果 z 的均方差。

训练问题：采用完整的 KGS 棋局数据训练该网络导致过拟合，因为前后局势是强相

关的，仅差一子。价值网络的训练结果是 MSE 为 0.19，而测试的结果是 0.37，这是因为这样训练的价值网络会记住 KGS 棋局的结果。

训练数据：为了解决这个问题，使用 p_ρ 自我博弈(self-play)生成的棋局，从不同棋局里采样出 3000 万不同局势，用这些数据训练价值网络。这样训练的结果是训练集上 MSE 为 0.226，而测试的结果是 0.234。

训练效果：$v_\theta(s)$一次就能判断出该局势下的胜率，比采用 p_ρ 的蒙特卡洛模拟少了 15 000 次的运算量。

阿尔法围棋使用的 4 个神经网络的特性与架构如表 3-1 所示。

表 3-1　阿尔法围棋使用的 4 个神经网络的特性与架构

深度神经网络	特性	网络架构	效果
π_{SL}	速度慢，准确性高。在 3000 万(s, a)上训练的随机监督学习策略	12 层网络，最后输出所有招数的概率分布	评估时间：3ms
$p_\pi(a\|s)$	速度快，准确性低。在 3000 万(s, a)上训练的随机监督学习策略	输入为所有模式特征的部分	评估时间：2μs
π_{RL}	随机强化学习策略，通过自我博弈训练	与 π_{SL} 相同	与 π_{SL} 博弈的胜率为 80%
$v_\theta(s)$	价值函数。从状态 s 使用 π_{RL} 获胜的胜率	与 π_{SL} 相同，但只输出胜率	比 π_{SL} 的计算量少

3.3.3　结合策略网络和价值网络的蒙特卡洛树搜索

结合策略网络和价值网络的蒙特卡洛树搜索如图 3-14 所示。

阿尔法围棋的蒙特卡洛树搜索方法如图 3-15 所示。

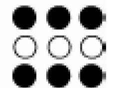
阿尔法围棋的蒙特卡洛树搜索有效结合了快速走子的策略网络 $p_\pi(a|s)$、策略网络 $p_\sigma(a|s)$和价值网络的 $v_\theta(s)$。

蒙特卡洛树遍历时，每条边(s, a)上会关联一个动作值 $Q(s, a)$、访问计数值 $N_v(s, a)$和先验概率 $P(s, a)$。

从根状态开始，通过仿真遍历整个树。在仿真的每个时间步 t，从状态 s_t选择动作 a_t，其中

$$a_t = \underset{a}{\operatorname{argmax}}(Q(s_t, a) + u(s_t, a))$$

最大化动作值再加上一个奖励 $u(s, a)$，其中



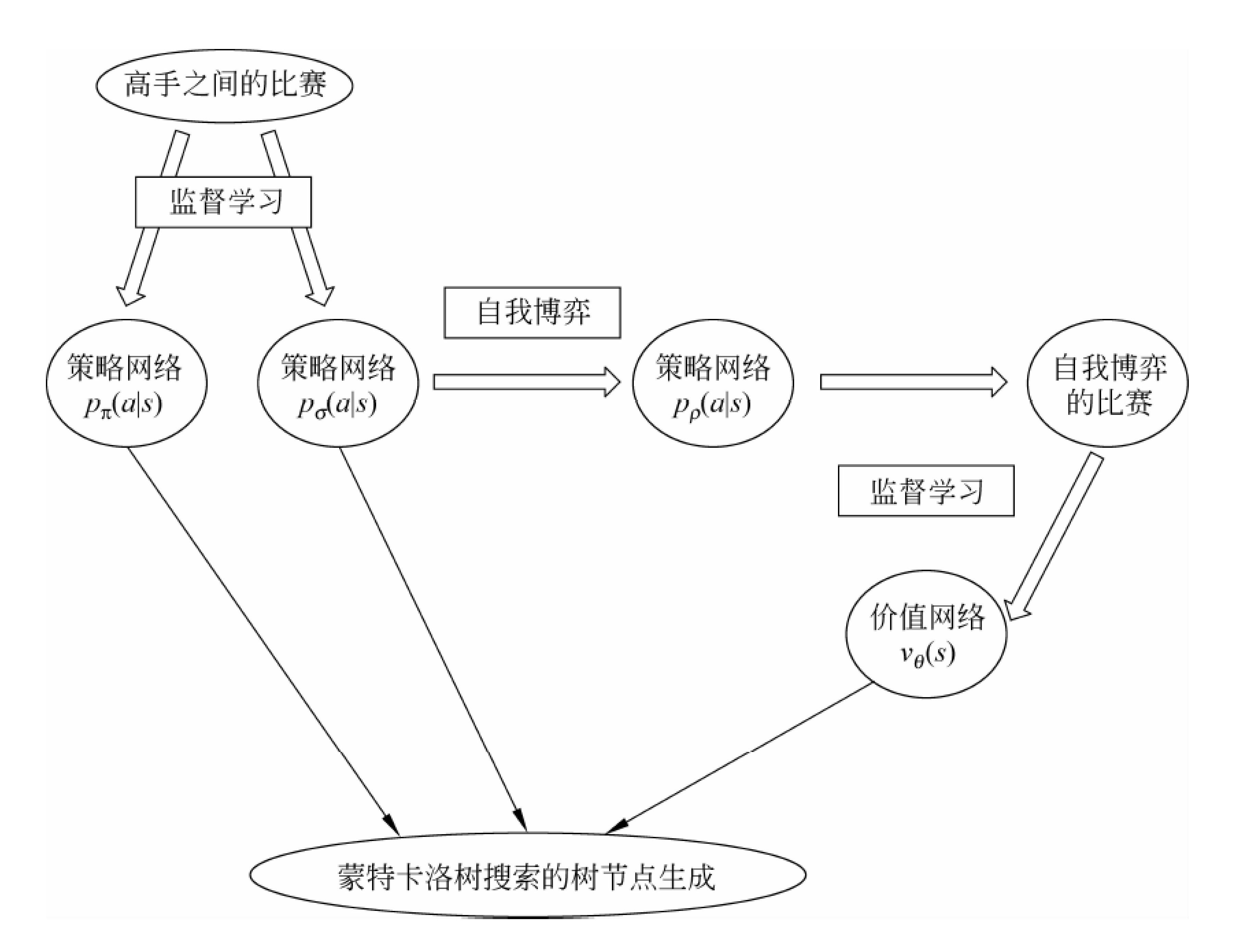


图 3-14　结合策略网络和价值网络的蒙特卡洛树搜索

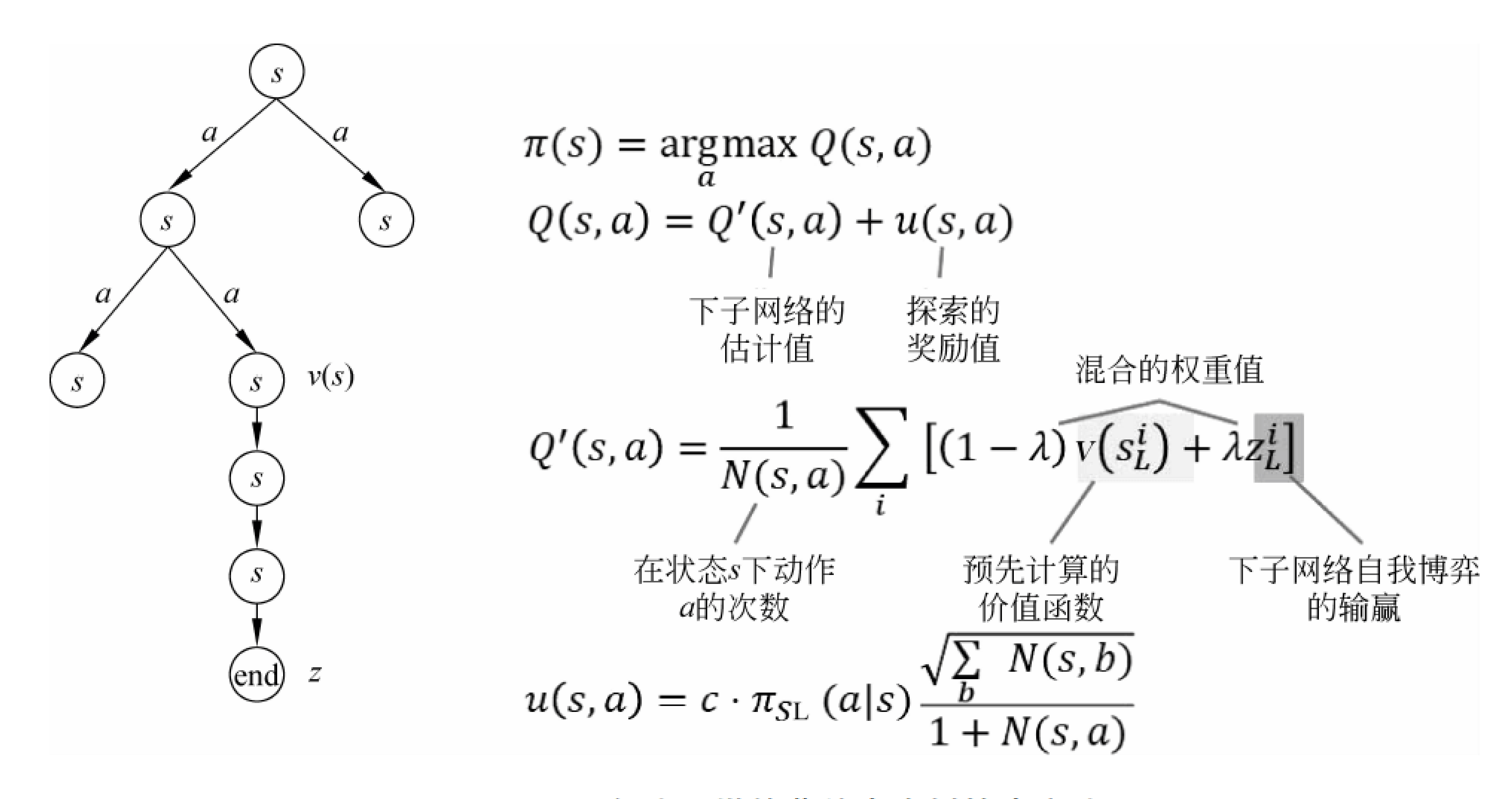


图 3-15　阿尔法围棋的蒙特卡洛树搜索方法

$$u(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$$

该奖励正比于先验概率，但是随着访问次数的增加而衰减，这样有助于去探索。

当树遍历到叶节点 s_L 时(在第 L 步)时，使用策略网络 $p_\sigma(a|s)$展开叶节点 s_L。输出的概率作为先验概率 $P(s|a)$。该叶节点评估采用两种方式：一种采用价值网络 $v_\theta(s_L)$；另外一种采用随机策略网络 $p_\pi(a|s)$。

最后该叶节点的 Q 值采用两种方法的折中，即 $\lambda=0.5$。

$$v(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L$$

这时会发现，采用 $p_\sigma(a|s)$比强化学习生成的 $p_\rho(a|s)$效果要好，原因是监督学习网络是基于人训练的，更符合人下棋的多样性特点。而价值网络的生成，采用强化学习比采用监督学习方法要好。

获取策略网络和价值网络的评估结果的计算量，比蒙特卡洛树搜索的计算量高几个数量级。为了有效地组合蒙特卡洛树搜索和深度神经网络，阿尔法围棋在通用处理器 CPU 上，使用异步多线程进行搜索树的搜索仿真；而在 GPU 上，通过并行计算获得策略网络和价值网络的评估结果。最终的阿尔法围棋用了 40 个搜索线程、48 个 CPU 和 8 个 GPU。并行多机的阿尔法围棋版本使用 40 个搜索线程、1202 个 CPU 和 176 个 GPU。

实际的阿尔法围棋采用 APV-MCTS(Asynchronous Policy and Value MCTS)方法，具体实现参考阿尔法围棋的 *Nature* 论文。

3.3.4 阿尔法围棋技术总结

通过前面介绍可看到阿尔法围棋综合使用了计算机围棋的各项技术：蒙特卡洛树搜索、强化学习和深度学习技术。正是采用了深度学习的深度卷积网络来模拟下棋，以及对局势的判断，才取得了计算机围棋上棋力的重大突破。这里，作者系统总结整理了阿尔法围棋的各项技术，包括阿尔法围棋继承的现有技术、新发展的技术以及具体的技术创新。

阿尔法围棋技术创新大全如表 3-2 所示。

在系统实现上，阿尔法围棋团队也做了不少创新。例如，做了分布式计算上的创新，解决了性能问题。

(1) 使用 Virtual Loss 来增加线程间的多样性。

(2) 控制蒙特卡洛树的更新，使得不同速度的计算过程能互相匹配。

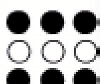


表 3-2　阿尔法围棋技术创新大全

继承现有技术	新增发展技术	技术创新
① 蒙特卡洛树搜索 ② 神经网络技术	使用深度神经网络： ① 从棋谱中训练快速走子网络； ② 从棋谱中训练 SL 网络； ③ SL 网络通过自我博弈训练得到 RL 网络； ④ 用 RL 网络得到价值网络	① 使用快速走子网络作为默认策略； ② 在树策略的每个节点的值的计算上：使用了 SL 网络的结果作为探索奖励项的初始值； ③ 价值网络的结果与快速走子的结果对半混合作为每次模拟的产出

3.4 小结

深度强化学习就是把强化学习(RL)和深度学习(DL)结合起来,用强化学习定义目标,用深度学习给出相应的机制,如 Q 学习等技术,以实现通用人工智能。目前强化学习得到了巨大发展,在很多领域,用强化学习方法可以实现人工智能的行为,例如阿尔法围棋。

阿尔法围棋的成功证明了用深度神经网络和蒙特卡洛树搜索方法,可以完成很多规则清晰的计算机游戏的人工智能,并能有效击败人类职业选手。阿尔法围棋的胜利,本质上是一拨搞计算机的人击败了另一拨职业围棋选手,也是人类创造智能工具和科技的胜利!

参考文献

[1] Richard S, Sutton. Andrew Barto, An Introduction to Reinforcement Learning[M]. Cambridge, The MIT Press, 1998.

[2] Szepesvari C. Algorithms for Reinforcement Learning [M]. Synthesis Lectures on Artificial Intelligence and Machine Learning 4, no. 1, Morgan & Claypool, 2010,1-103.

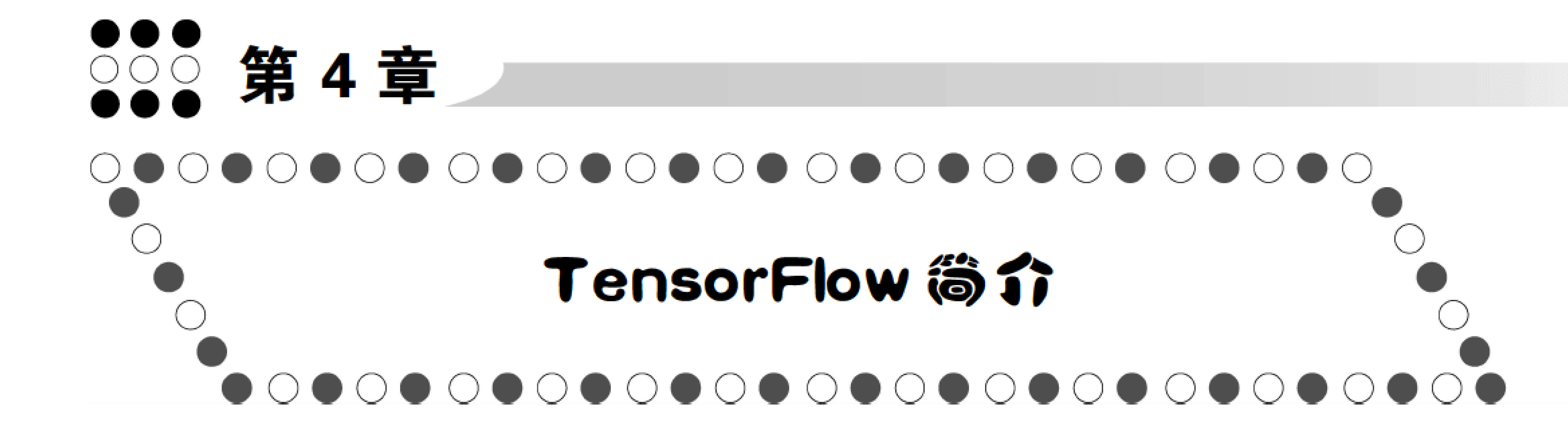
[3] Williams R J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning[M]//Reinforcement Learning. Springer US, 1992, 229-256.

[4] Christopher C, Amos S. Training Deep Convolutional Neural Networks to Play Go[C]. ICML-15, 2015, 1766-1774.

[5] Maddison C J, Huang A, Sutskever I, et al. Move Evaluation in Go Using Deep Convolutional

Neural Networks[J]. Computer Science, 2015.
[6] Chaslot G, Chatriot L, Fiter C, et al. Combining Expert, Offline, Transient and Online Knowledge in Monte-Carlo Exploration[J]. Baptiste, 2010.
[7] Abramson B. Expected-Outcome: A General Model of Static Evaluation[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 1990, 12(2):182-193.

第 4 章



2015 年 11 月 9 日，谷歌公司开源了大规模机器学习系统 TensorFlow，大大普及了深度学习的应用。最初是由谷歌公司的机器智能研究部门的谷歌大脑团队开发了 TensorFlow，TensorFlow 是继第一代机器学习系统 DistBelief 之后推出的第二代机器学习系统，主要目的是为进行机器学习和深度神经网络研究。

但是，谷歌大脑开发团队开发 TensorFlow 时，其目标就要求这套工具系统有足够的通用性和易用性，适用于多种不同的机器学习领域，能够运行在不同的软件和硬件平台上。

2017 年 2 月 15 日，谷歌大脑团队在 TensorFlow-Dev-Summit 上，正式推出了 TensorFlow 1.0 版本。目前在微软 Azure 云、谷歌云上的 TensorFlow 标配版本为 1.3。

4.1 TensorFlow

Tensor(张量)意味着 *N* 维数组。1 维时就是向量，2 维时就是矩阵；通过图像可以代表更高维的数据流，例如，图像可以用三维张量(行，列，颜色)来表示。Flow(流)意味着基于数据流图的计算。有许多运算(图中的节点)应用在数据流上。张量从图的一端流动到另一端，这就是 TensorFlow(张量流)。其中，“边”代表张量(数据)，节点代表运算处理。

TensorFlow 是用数据流图(Data Flow Graph)进行数值计算的一个开源软件库。图中的节点表示数学运算，而图的边代表它们之间传送的多维数据阵列(张量)。灵活的架

构可以使用同一个 API 的调用，部署在一个或多个 CPU 或 GPU 的桌面计算机、服务器或移动设备。

图 4-1 给出 TensorFlow 的分层架构。

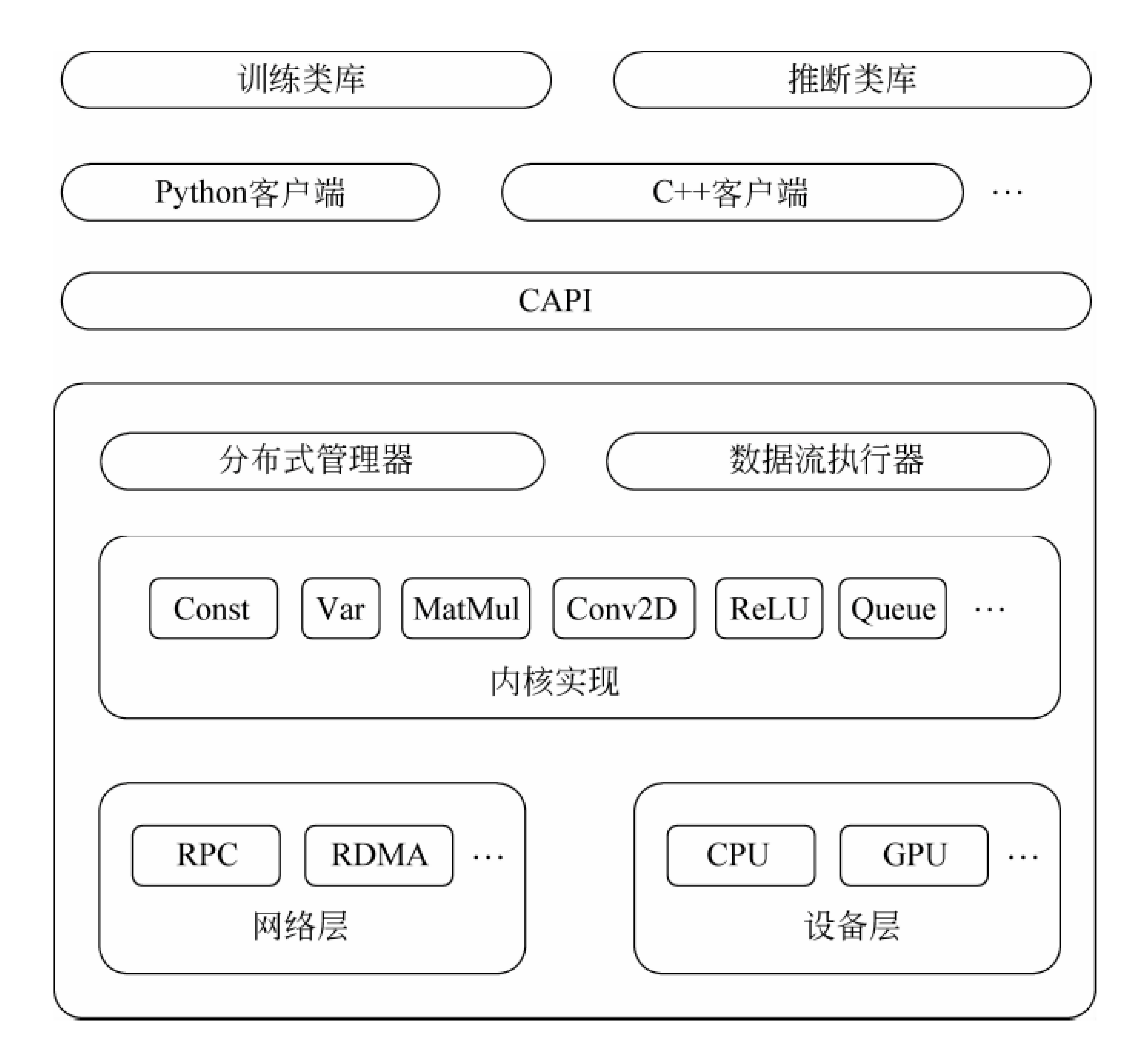


图 4-1　TensorFlow 的分层架构

图 4-2 给出了 TensorFlow 用于模型训练过程的数据流图。

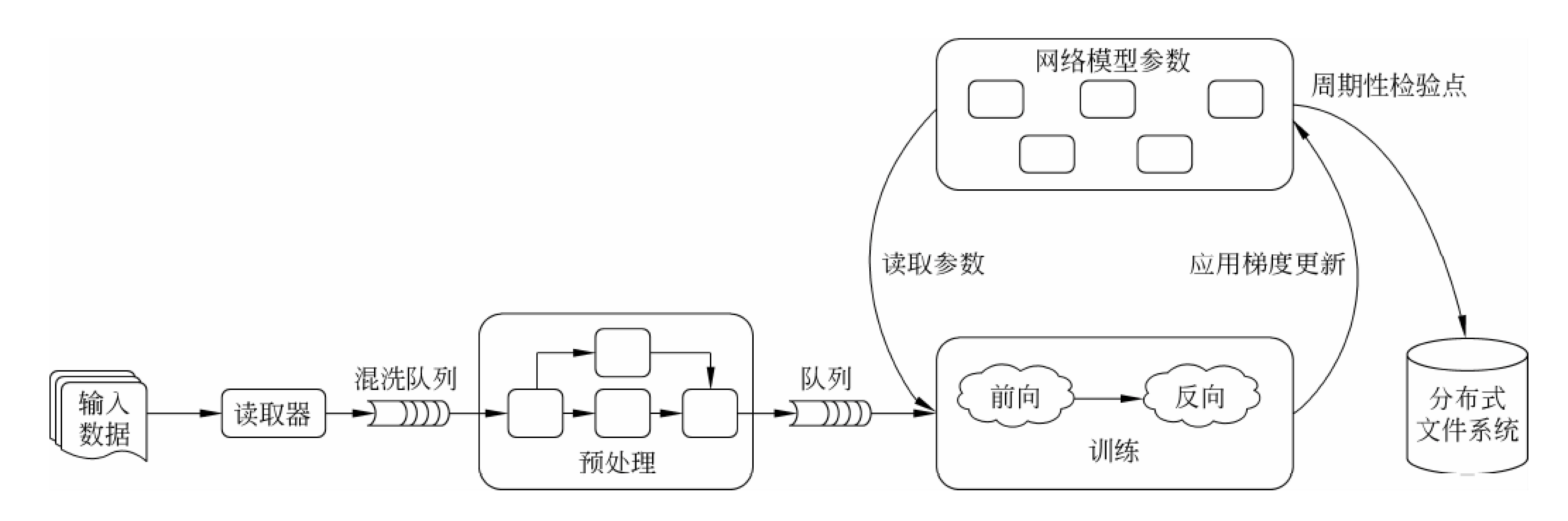


图 4-2　TeensorFlow 的数据流图示例

4.2 TensorFlow 使用

4.2.1 TensorFlow 起步

TensorFlow 的核心代码是用 C++ 编程，前端用 Python 或者 C++。TensorFlow 使用 Python 的例子如下。

```
$ python
>>>import tensorflow as tf
>>>hello =tf.constant('Hello, TensorFlow! ')
>>>sess =tf.Session()
>>>sess.run(hello)
Hello,TensorFlow!
>>>a =tf.constant(15)
>>>b =tf.constant(12)
>>>sess.run(a+b)
17
>>>
```

4.2.2 TensorFlow 数据的结构

TensorFlow 参考了 Python 中的 Numpy 库的很多概念，如 Array 的概念、Shape 的概念、reduce-sum 函数、reshape 函数、argmax 函数。

TensorFlow 中，Tensor 可以理解为 Numpy 中的 Array。当然两者定位不同。

两个“世界”：TensorFlow ⇔ Numpy

Tensor ⇔ Array

Array 是在内存中切实存在的数据（矩阵），有形（shape）有数据（data）。

Shape 是一个 Array 的大小，例如一个三行四列的矩阵，Shape 是[3，4]。更高阶的 Array 其 Shape 会更长。

Tensor 是抽象意义的数据，只有形没有数据。只有在 Session.run()的时候，Tensor 才真正赋予数值，并开始迭代运算。

4.2.3 TensorFlow 的工作流程

1. 构建网络结构(计算流程)

预留用于数据输入、参数配置的锚点。

2. 填充锚点,运行网络

(1) Tensor 存在于网络结构中,只有形没有内容。

(2) Tensor 的运行结果是 Array。

其内容根源上来自于锚点被填充的数据。

4.3 Tensor 运算

通过 Tensor 进行计算操作得到的结果仍然是 Tensor。Tensor 参与的计算操作,其操作只是个标记,并没有真正使用计算资源进行实际运算。例如,2×4 矩阵与 4×3 矩阵相乘结果必然是 2×3 矩阵,并不需要知道矩阵的具体内容。

```
tc =tf.matmul…

tc =ta +tb
```

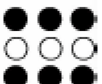
Tensor 参与的操作,其计算一般只是在 Shape 的基础上进行的,只有 run 操作除外。TensorFlow 通过 run 操作完成 Tensor 的实体化(数据化),在 run 操作时指定需要实体化的 Tensor 列表。

```
a, b, c, … =sess.run([ta, tb, tc …], feed_dict ={…})
```

Tensor 与 Tensor 之间有依赖关系,为了计算 A,必须先获得 B 的结果,这就是依赖。TensorFlow 会分析依赖关系,执行最少的计算以得结果。

若 A−>B−>C−>D,则 run([B]) 只会执行 A 和 B。

一个全连接网络 Dense 的内部依赖关系如图 4-3 所示。



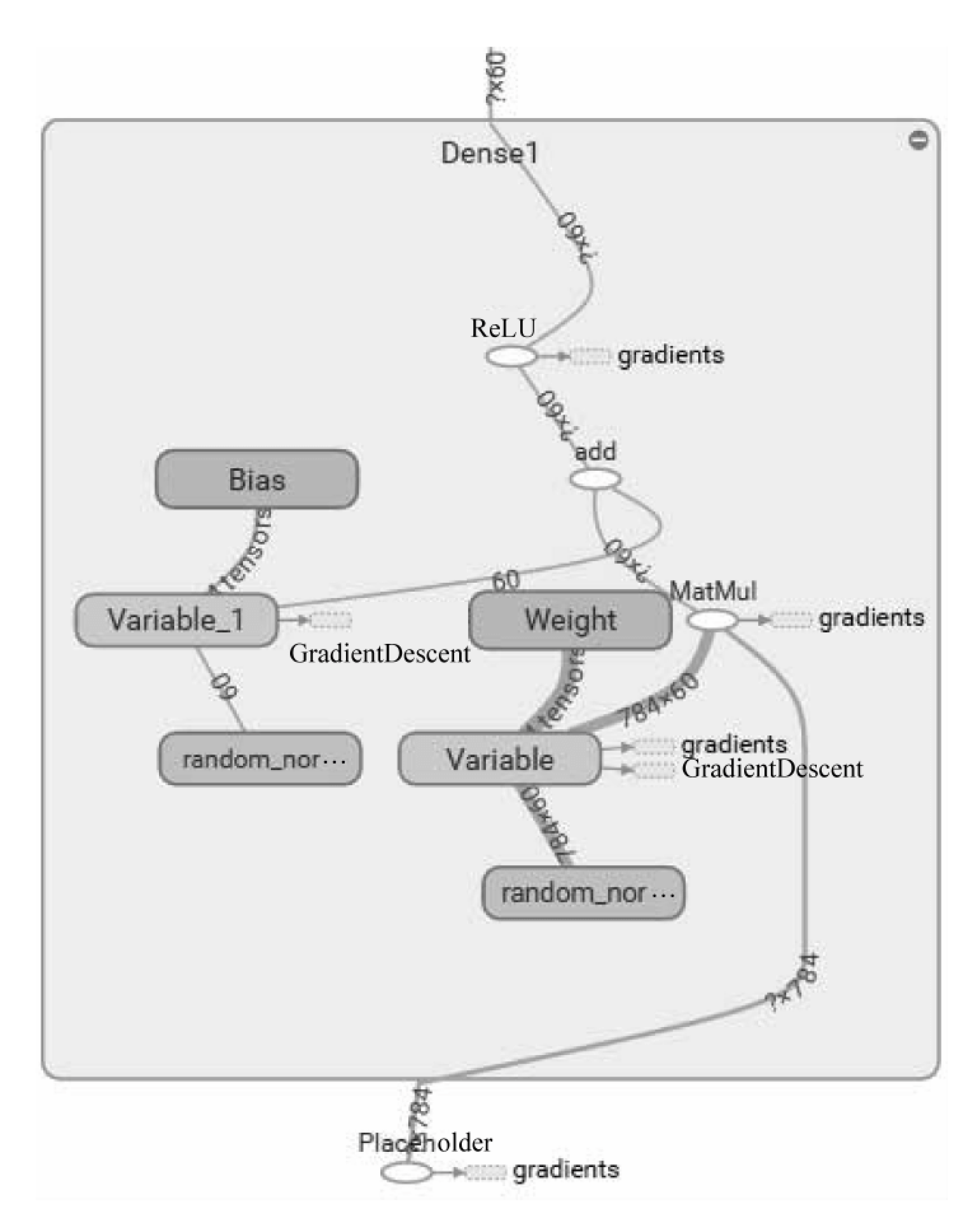


图 4-3　一个全连接网络 Dense 的内部依赖关系

4.4 导入实验数据

将数据导入 TensorFlow 程序的方法主要有 3 种。

1. 使用 Feeding 机制

Feeding 机制可以把数据注入到计算图中的任何一个 Tensor。在初始化时，通过提供数据给 run 函数和 eval 函数的 feed_dict 参数。

feeding 机制使用 Python 代码，将数据文件(如音频或图像文件等)转化为 Numpy

的 Array 数据，通过赋值给 run 函数的 feed_dict 参数中的某个张量，变为 TensorFlow 中的张量。

这种方法的核心问题是如何将数据文件的内容读取出来，转换为 Numpy 的 Array 对象。

2. 从文件中读取数据

在 TensorFlow 计算图的起始处，使用输入 pipeline 从文件中读取。TensorFlow 自带的数据读取组件，通过一系列针对 Tensor 的文件操作、数据操作，获得未来将包含数据的 Tensor。只有被计算后才会产生真正的操作，否则只是在做一些标记。

3. 预加载数据

TensorFlow 计算图的参数和变量预先存放了所有的数据(这种方法适用于加载小的数据集)。

4.4.1 NumpyArray 方法

1. 图片

从 PIL 读取图片，将 PIL 的图片对象直接给 Numpy 的构造函数，直接得到 Numpy 的 Array 对象。

后期处理：数据格式转换、数组变为矩阵等。

2. 二进制

读取 string 并转为 bytes：

```
buf =bytes(f.read())
```

将 bytes 按照特定格式，解释为一维的 Numpy 的 Array 对象：

```
floats =numpy.frombuffer(buf, dtype= numpy.float32)
```

将数组变为矩阵：

```
floats =numpy.reshape(floats[3:],[freq, time], order='C')
```

4.4.2 TensorFlow 组件方法

1. 获得文件列表(list)

选择 1：Python 中的 list(自己在 Python 范畴内搞定)。

选择 2：TensorFlow 的组件，glob_string 是类似"./*.jpg"这种可以带通配符的字符串。

```
filenames =tf.train.match_filenames_once(glob_string)
```

它只是个标记，并没有真正地进行匹配。

2. 将文件列表传给 input_producer

```
input_producer =tf.train.string_input_producer(
        filenames,
        shuffle=true,
        capacity=config.PRODUCER_CAPACITY
)
```

3. 用 reader 从 input_producer 中读取文件名(key)和文件内容(value)

```
reader =tf.WholeFileReader()
key, value =reader.read(input_producer)
```

4. 对读取的内容进行解码，有许多默认的解码器和通用解码器

```
flt_seq =tf.decode_raw(content, out_type=tf.float32, little_endian=True)
```

decode_raw 和 Numpy 的 frombuffer 类似。

5. 得到的是一个数组，可以用 slice、reshape 等操作进一步将其塑造成需要的结果

```
img =tf.reshape(tf.slice(flt_seq, [3], [total]),
               [config.IMAGE_HEIGHT, config.IMAGE_WIDTH, config.IMAGE_CHANNEL])
```

至此 img 就是一个 3 阶的 Tensor 了。

4.4.3 TensorFlow 示例

Tensor 只存在于网络结构中，只有形没有内容。Tensor 的运行结果是 Array，其运行对象来自于锚点的被填充数据。下面用手写 0～9 阿拉伯数字的 MNIST 数据集的 TensorFlow 实验过程，来说明 TensorFlow 的计算过程。

MNIST 数据集包含 70 000 张手写数字图片，每张图片大小是 28 像素×28 像素。MNIST 数据被分为 3 部分，其中 55 000 张用于训练（MNIST. Train），10 000 张用于测试（MNIST. Test），5 000 张用于验证数据（MNIST. Validation）。实验采用 TensorFlow 框架来建模和训练一个 3 层神经网络，完成这些图片的自动识别过程。实验用的源代码见本章的参考文献[3]。

首先，构建网络结构（计算流程），预留用于数据输入、参数配置的锚点。

```
x =tf.placeholder(tf.float32, shape=[None, 784]) #数据锚点，注意 shape
y_ =tf.placeholder(tf.float32, shape=[None, 10])#标签锚点，None 表示未知，任意
#标签长度是 10，表示使用 one_hot 方式表示标签(10 个数字中只有一个是 1，其他是 0)

w =tf.Variable(tf.zeros([784,10])) #声明网络的参数变量
b =tf.Variable(tf.zeros([10]))

y =tf.matmul(x,W) +b #简单的直线拟合问题，输出预测标签 y

#使用交叉熵作为输出
cross_entropy =tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y, y_))

#使用优化算法最小化交叉熵
train_step =tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

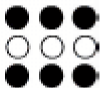
其次，从官方获得 MNIST 的 Array 数据，填充锚点，运行计算图。

```
mnist =input_data.read_data_sets('MNIST_data', one_hot=True)
```

使用自己的标签数据，形式如下：

```
data, label =myownfunction() #data、label 的 shape 要和锚点匹配
```

运行训练网络。



```
with tf.Session() as sess:
sess.run(tf.global_variables_initializer())  #初始化所有变量
for i in xrange(0, N): #迭代
batch_data, batch_label =myfunction() #获得用于迭代的一批数据
feed_dict ={x: batch_data, y_: batch_label} #配置好锚点的输入

#传入锚点配置,运行训练算符和交叉熵算符
_, loss =sess.run([train_step, cross_entropy], feed_dict=feed_dict)
```

train_step 的返回被丢弃，cross_entropy 的返回是 loss 值。

sess.run()函数的参数列表中，人们可以设定希望计算的 Tensor，从而在返回值中获得它们。这里，可以添加一个计算精确度(accuracy)的 Tensor：

```
correct_prediction =tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy =tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

在运行时，将精确度加入参数列表。

值得注意的是，TensorFlow 会使用尽可能少的计算，来获得输入 Tensor 的计算结果。例如：

```
b =a +1 ;c =b+1
```

这时，sess.run([b])会计算 b 和 a 的值，而不会计算 c 的值。因为 b 的计算不需要 c 的参与。

4.5 TensorBoard 示例

TensorBoard 是 TensorFlow 自带的图形化工具，用于图形化展示神经网络训练的效果。图形化展示的内容包括网络结构、网络内部的权重和偏置参数的分布，以及每一层的输入结果和最终的输出结果。

我们还是使用 MNIST 数据集的 TensorFlow 实验，训练手写数字图片识别的神经网络。将训练的结果用 TensorBoard 展示，这需要补充一些操作。

在 model.py 中，增加了 summary 的内容。只有这样，才能通过 TensorBoard 展示。

```
summary_op =tf.summary.merge_all()

#writer for summary
        writer =tf.summary.FileWriter('./log/', sess.graph)

        _,win_rate, summary =sess.run([train, wrate, summary_op], {X:datax, Y:
datay, lr:learning_rate})

writer.add_summary(summary, i)
```

1. 从 Gitlab 上下载源码

使用 git 或者下载压缩包，将代码放到目录/tfExample/下。

2. 解压数据

/tfExample/data 下有压缩过的数据，分 3 个文件存储，打开 mnist.zip，将其中的 mnist.mat 解压出来，使得存在文件/tfExample/data/mnist.mat。

3. 运行训练过程

运行命令：

```
$ python./tfExample/model.py
```

该过程会生成两个文件夹：一个是/tfExample/log，另一个是/tfExample/metaEmbed。

4. 打开 TensorBoard

运行命令：

```
$ python -mtensorflow.tensorboard --logdir=./tfExample/log/
```

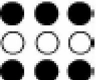
该命令会在本机的 6006 端口打开网页服务，从而可以从网页端看到训练进程中的各种参数和信息。

5. 打开浏览器

在地址栏中输入：

```
http://localhost:6006 (或 IP 地址:6006)
```

TensorBoard 展示效果如图 4-4 所示。



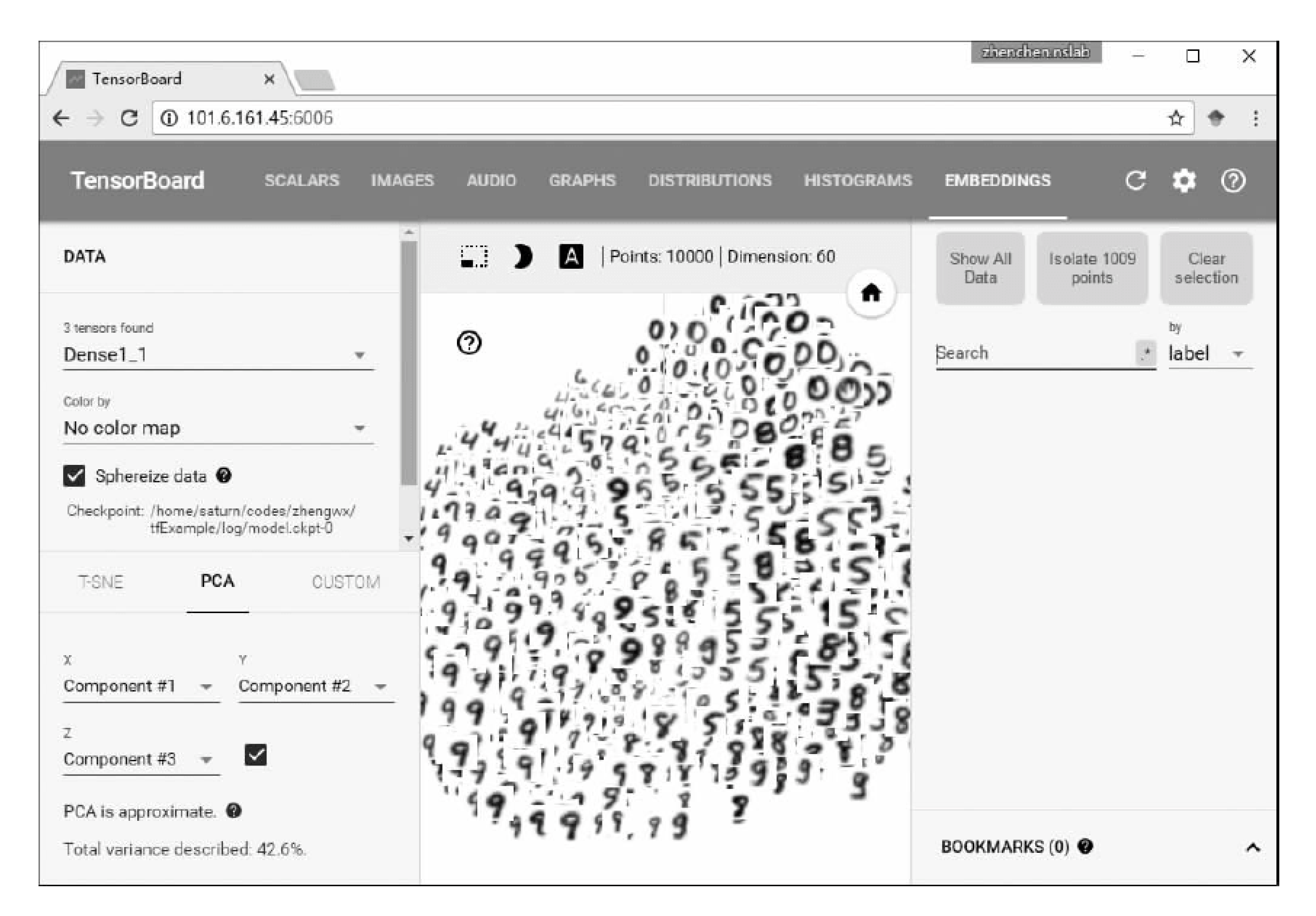


图 4-4　TensorBoard 展示 MNIST 数据集训练效果

4.6 小结

本章介绍了深度学习的程序框架 TensorFlow，以及如何使用 TensorFlow 导入数据处理，并给出一个 MNIST 数据的处理示例。最后，介绍 TensorFlow 自带的 TensorBoard 图形化界面。TensorBoard 可以直观地展示网络的训练效果、内部的权重分布等重要信息，深入理解网络的演化过程，以及最终的训练效果。

参考文献

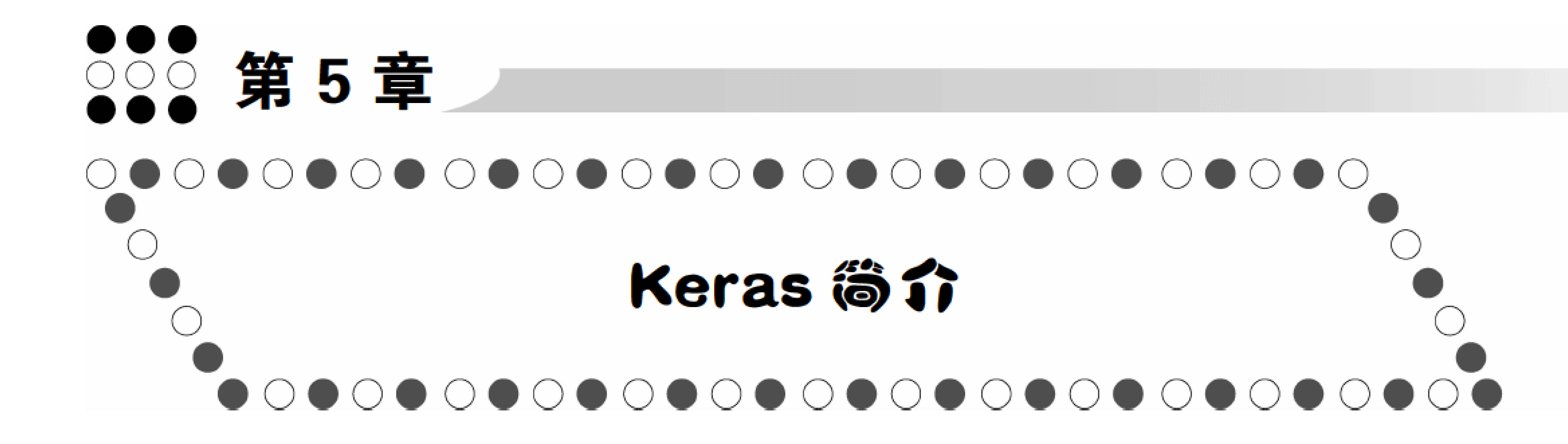
[1] TensorFlow[EB/OL]. http://www.tensorflow.org.

[2] Abadi M, Agarwal A, Barham P, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems[C]. OSDI2016.

[3] tfExample[EB/OL]. http://gitlab.icenter.tsinghua.edu.cn/saturnlab/tfExample.

[4] TensorFlow Architecture[EB/OL]. https://www.tensorflow.org/extend/architecture.

[5] Jeff Dean, et al. Large Scale Distributed Deep Networks[C]. nips 2012.



第 5 章 Keras 简介

Keras 是由谷歌公司的研究员 Francois Chollet 开发的一套高层框架，进一步封装 TensorFlow。TensorFlow 1.0 版本已经集成了 Keras 框架。目前 Keras 版本为 2.0。

5.1 Keras

Keras 是对 TensorFlow、Theano 和 CNTK 的进一步封装，抽取共性，二选一。Keras 设计采用极简主义原则，是一套高度模块化的神经网络架构库。Keras 具有方便使用的特点，如简洁的网络定义方式，常用技巧的封装。同时，Keras 又保留足够的扩展性，可以自行实现各种层(Layers)。

机器学习模型、训练、预测过程如图 5-1 所示。

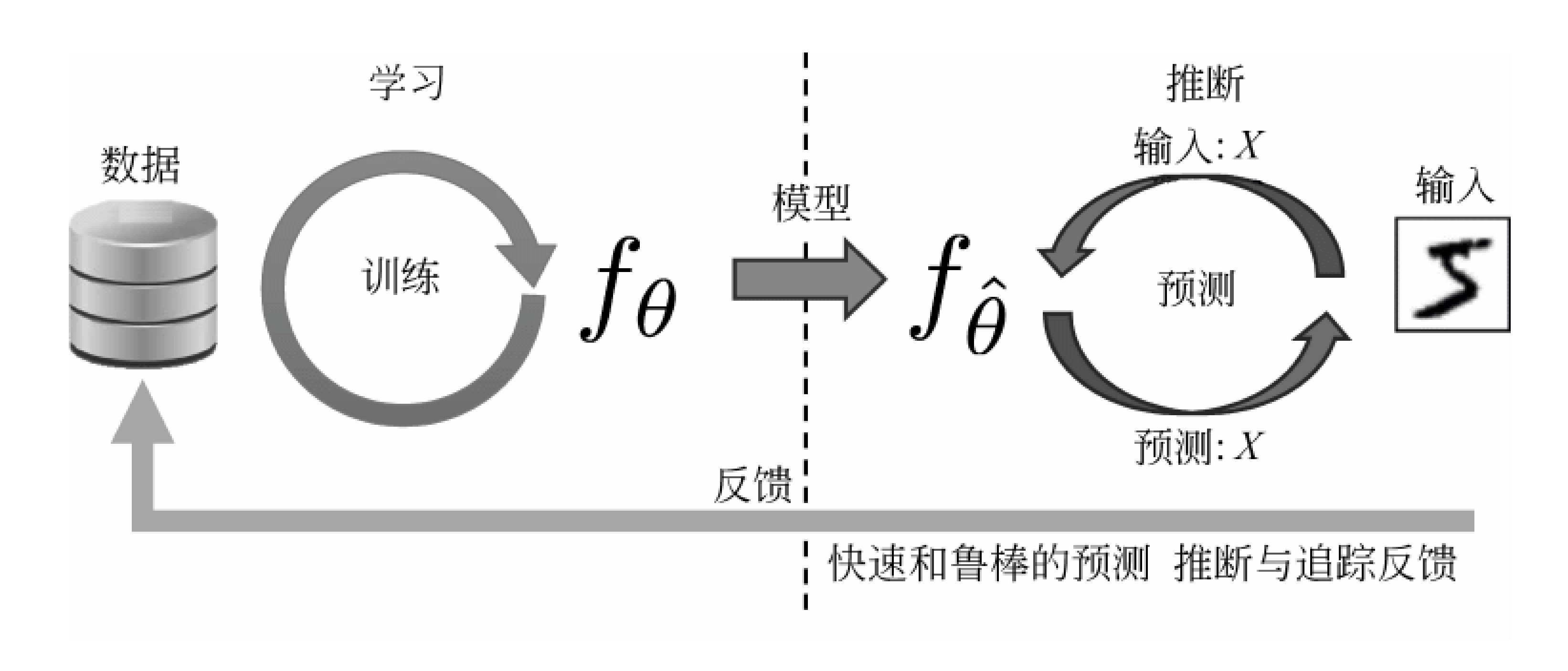
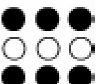
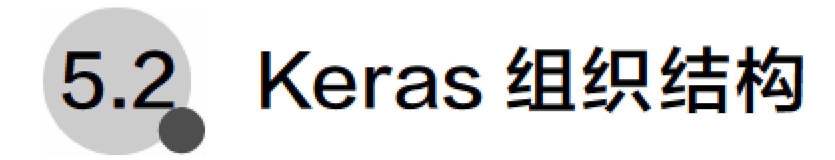


图 5-1 机器学习模型、训练、预测过程





5.2 Keras 组织结构

5.2.1 Models

Keras 核心的数据结构称为 Model，如图 5-2 所示，Model 是组织层的一种方式。最简单的 Model 类型是序列模型（Sequential Model）。序列模型是指串接很多层的管道。

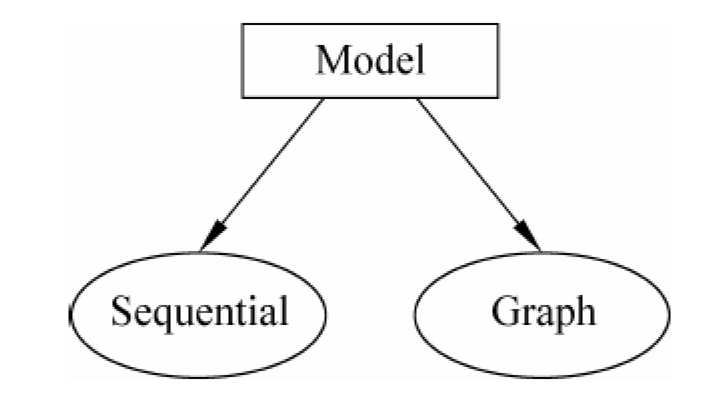


图 5-2　Keras 核心的数据结构 Model

5.2.2 Core Layers

Keras 核心层（Core Layers）包括 Dense、Activation、Dropout、Flatten、Reshape、Permute、Repeat Vector、Lambda、Activity Regularization、Masking。

5.2.3 Layers

Keras 非核心层包括 Convolutional Layers、Pooling Layers、Locally-connected Layers、Recurrent Layers、Embedding Layers、Merge Layers、Advanced Activations Layers、Normalization Layers、Noise Layers。

5.2.4 Activations

Keras 激活函数（Activations）包括 softmax、elu、softplus、softsign、relu、tanh、sigmoid、hard_sigmoid、linear 等，函数表示式如下：

```
softmax(x), elu(x, alpha=1.0), softplus(x), softsign(x), relu(x, alpha=0.0,
max_value=None), tanh(x), sigmoid(x), hard_sigmoid(x), linear(x)
```

5.2.5 Optimizers

Keras 优化器(Optimizers)包括 SGD、RMSprop、Adagrad、Adadelta、Adam、Adamax、Nadam、Optimizer。

Nadam 采用 Nesterov 加速梯度算法。基本思想是模型的参数更新是基于之前的所有梯度(全局的信息),而不是仅基于当前的梯度(局部信息),训练收敛的速度是可以进一步提升的。

RMSprop、Adagrad、Adam 都是自适应调整学习率的算法。

5.3 Keras 实践

5.3.1 Keras 安装

首先,安装 Keras 要求的后端软件,如 Theano、TensorFlow、CNTK 等。这里以 TensorFlow 为例,可访问 https://www.tensorflow.org/install/。

选择安装 TensorFlow 的操作系统环境,如 Linux、MacOS、Windows 等。

如果需要 GPU 的支持,则不能默认安装,从网页给出的一张列表中,选择合适的 pip 包。

在 Linux 环境下,安装 Keras 命令如下:

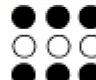
```
$ sudo apt-get install libhdf5-10 libhdf5-dev
```

在 Windows 环境下,安装 Keras 命令如下:

```
$ pip install keras numpy scipy pyyaml h5py
```

至此,Keras 安装完毕。

如果在安装运行中,系统报缺失 xxx module 的错误,则说明需要额外的程序包支持的功能。此时,需要自己寻找缺少的包是什么,再逐一安装。



5.3.2 Keras 使用

1. 定义网络

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model =Sequential([
    Dense(32,input_dim=784),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

网络结构是以一个栈的形式给出，一层接一层。

第一层的 input_shape/input_dim 参数指定输入层的大小。例如，这里指定一个 784 维的输入。

input_dim 总是存在于第一层的参数列表，而与第一层是什么类型的网络无关。

最后一层一般是 softmax 作为输出层。此例中给出了 10 个输出，即表示该网络解决 10 类分类的问题。

2. 编译网络

给定网络的训练目标，给定网络的优化算法如下：

```
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

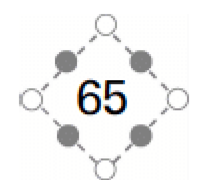
optimizer 是指使用 rmsprop 算法进行梯度下降。

loss 是指网络使用多类交叉熵作为优化目标。

metrics 是指在训练过程中，我们希望观察到准确度这个指标是如何变化的。

3. 训练网络

```
import numpy as np
data =np.random.random((1000, 784))
```



```
labels =np.random.randint(2, size= (1000, 1))

#train the model, iterating on the data in batches
#of 32 samples
model.fit(data, labels,nb_epoch=10, batch_size=32)
```

上述代码产生 1000 个符合 input_shape 的数据和标签，然后传给 model. fit，从而开始迭代训练过程。

训练过程会动态输出损失(loss)和准确度(accuracy)等参数。

4. 评估网络和数据预测

代码如下：

```
model.evaluate(self, x, y, batch_size=32, verbose=1, sample_weight=None)
model.predict(self, x, batch_size=32, verbose=0)
```

5. 保存/载入网络

将保存/载入网络结构与参数，采用 hdf5 格式保存网络参数数据，代码如下：

```
from keras.models import load_model
model.save('./model.h5')
model2 =load_model('./model.h5')
```

5.4 小结

本章介绍了 Keras 的基本概念及安装和使用。Keras 通过封装 TensorFlow，大大提高了 TensorFlow 的易用性。

参考文献

[1] Keras: Deep Learning Library for Theano and TensorFlow[EB/OL]. http://keras. io.
[2] Keras Source Code[EB/OL]. https://github. com/fchollet/keras.
[3] Keras 中文文档[EB/OL]. https://keras-cn. readthedocs. io.



第 6 章



声控智能 1——预处理与训练

6.1 声控智能

语音控制功能是指通过语音，给手机发出语音指令，手机 APP 自动识别指令种类，进行相应的控制。下面介绍一个用语音控制蓝牙音响的智能硬件的例子。

6.1.1 语音指令

该蓝牙音响的语音指令规定如下。其中字母为录音的文件的名称。

```
a:蓝牙开机
b:蓝牙拨打电话/bb:蓝牙打电话
c:蓝牙接听电话/cc:蓝牙接电话
d:蓝牙拒接
e:蓝牙播放音乐/ee:蓝牙开始音乐
f:蓝牙暂停音乐/ff:蓝牙停止音乐
g:蓝牙上一首/gg:蓝牙上一曲
h:蓝牙下一首/hh:蓝牙下一曲
i:蓝牙音量增大/ii:蓝牙声音增大/iii:蓝牙音量增加/iiii:蓝牙声音增加
j:蓝牙音量减小/jj:蓝牙声音减小
k:蓝牙关机
l:蓝牙电量提醒/ll:蓝牙还剩多少电/lll:蓝牙还剩多少电量
```

6.1.2 语音时频谱图

时频谱图是对语音文件进行短时傅里叶变换(STFFT)获得的。变换以后,更能体现语音的特征,便于后端处理。一个典型的语音时频谱图如图 6-1 所示。

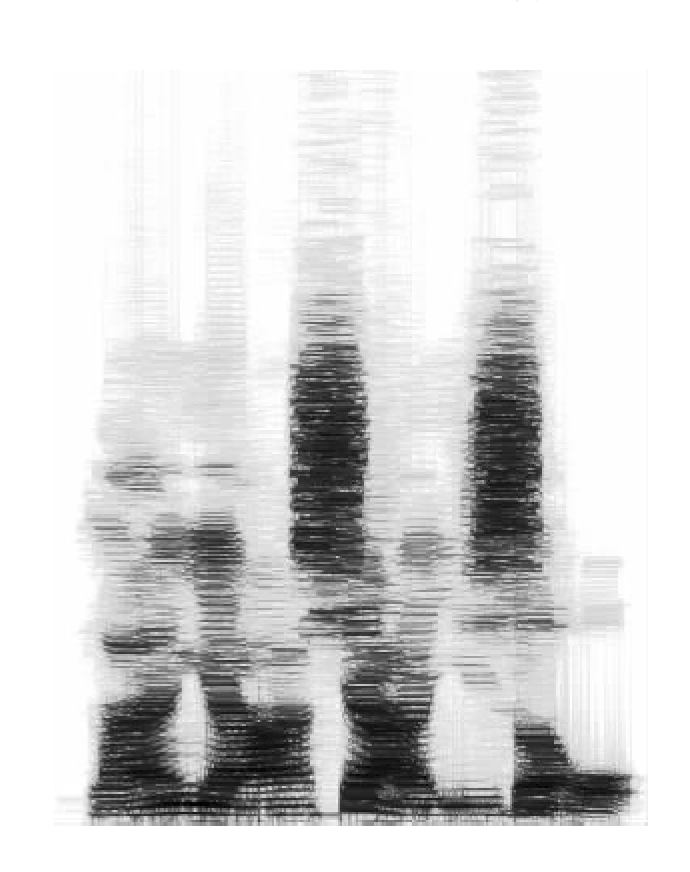


图 6-1 语音时频谱

6.1.3 语音文件录音

用笔记本或者手机录下以上 24 条语音指令,生成对应指令的. mp3 文件,文件名即为指令名字编号,如图 6-2 所示。

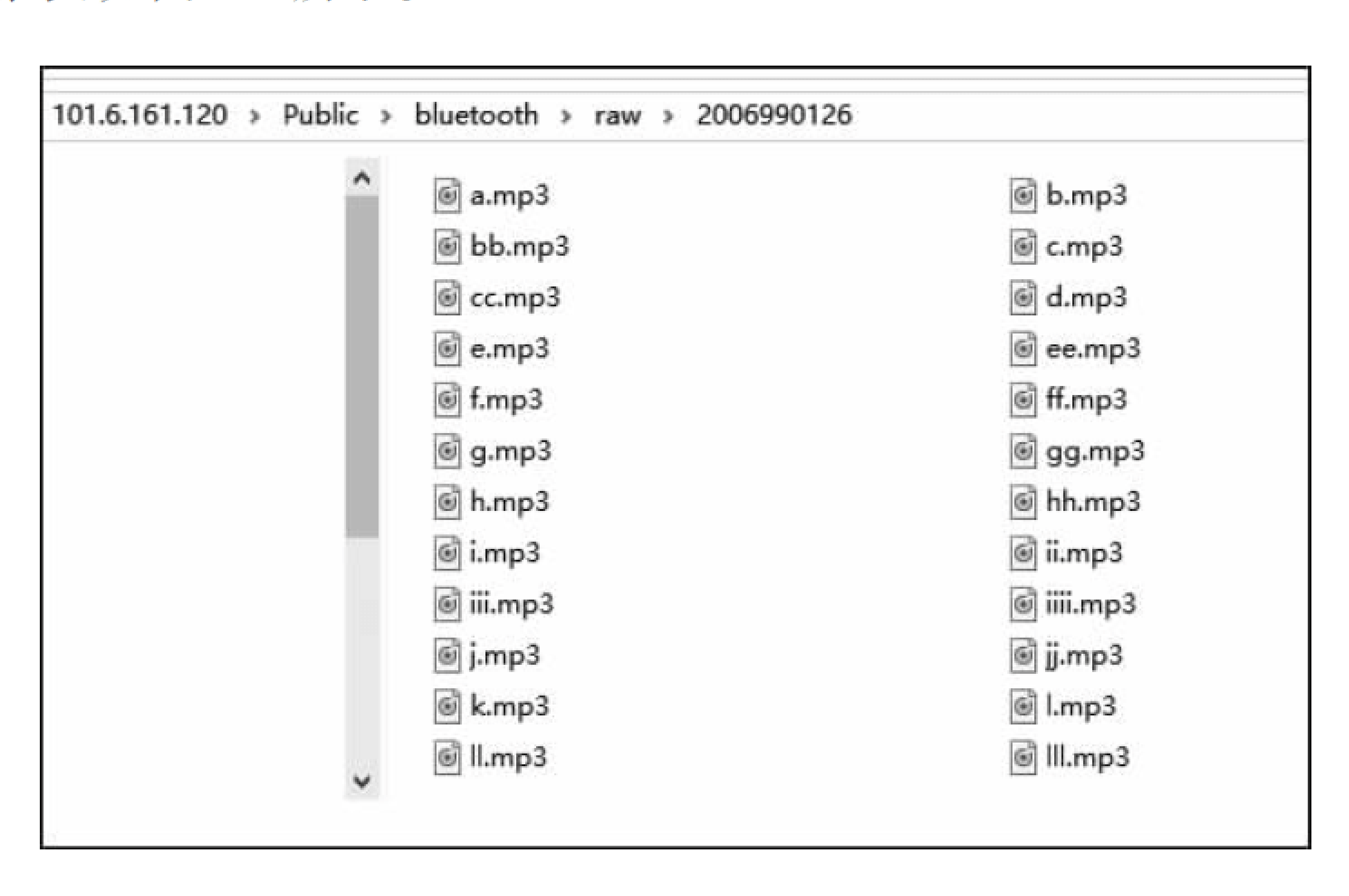
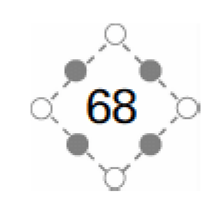
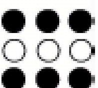


图 6-2 语音指令生成录音文件





6.2 实验过程

6.2.1 语音数据预处理

录音文件因为来源于不同的地方，录音格式有 *.m4a、*.wav、*.mp3、*.mp4、*.3gpp 等，需要进行统一处理。

Spectrum 项目源代码用来处理不同的各种格式的录音文件，并转换为统一的 *.wav 格式，生成语音频谱图。实验前先下载 Spectrum 代码，下载地址为 http://gitlab.icenter.tsinghua.edu.cn/saturnlab/spectrum。

这里，Spectrum 项目用 ffmpeg 工具把各种格式的录音文件转换为统一的 *.wav 格式的数据，便于统一处理。

ffmpeg 程序是一套跨平台的音视频媒体的录制、格式转换和流媒体的软件，下载地址为 https://ffmpeg.org/。

1. Spectrum 项目编译

Spectrum 项目运行在 Linux 上，生成频谱图的整个流程还需要 ffmpeg 和 fftw3 工具的支持。安装 ffmpeg 的方法如下：

```
$ sudo apt-get update
$ sudo apt-get install ffmpeg
```

安装 fftw3 的方法如下：

```
$ sudo apt-get install libfftw3-dev
```

Spectrum 项目源代码是使用 scons 管理的，其中 scons 类似于 make 命令，而 SConscript.p 类似于 makefile 文件。scons 使用 Python 编写，所以之前需要安装好 Python。

安装 scons 的方法如下：

```
$sudo apt-get install scons
```

如果提示没有安装编译工具，则先安装使用编译工具：

```
$sudo apt-get install build-essential
```

2. 统一录音格式

Spectrum 项目的 convert_wav. sh 脚本完成统一格式的操作。

```
./res/pcmf32le/convert_wav.sh
```

Spectrum 项目用 ffmpeg 工具对单个录音文件进行处理。ffmpeg 程序生成标准 wav 文件的命令如下：

```
ffmpeg-inputfile-ac1-acodecpcm_f32le-af 44100 inputfile
```

convert_wav. sh 脚本对录音文件进行批量转换。该脚本会自动找到 input_dir 目录下的所有文件(包括子目录)，尝试使用 ffmpeg 命令进行格式转化。如果转换成功，则保存新格式的文件，否则失败则跳过。所有转换后的新格式文件会统一放置在 output_dir 目录下，该目录没有子目录。

convert_wav. sh 脚本使用的方法如下：

```
sh./convert_wav.sh ./input_dir output_dir
```

3. 时频谱图生成

Spectrum 项目的 convert_bin. sh 脚本完成批量生成时频谱图的操作。

```
./res/spectrum/convert_bin.sh
```

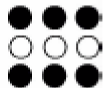
可以修改 convert_bin. sh 的一些选项来改变转化的参数，使用方法和 wav 转换类似，只是最后多加了一项指定程序路径的项，用法如下：

```
sh./convert_wav.sh./input_diroutput_dir path_to_program
```

Spectrum 项目在有 SConstruct 文件的目录下运行 scons 后，即可在 spectrum. p/bin 目录下找到可执行程序 Spectrum。将其作为脚本的第 3 个参数即可。

6.2.2 语音识别网络

接下来需要完成语音识别神经网络的设计方案。我们使用 Keras/TensorFlow 框架，完成语音识别神经网络的设计与实现。



语音识别神经网络可以采用两种类型的网络，即三层的 FCN 和 8 层的 CLDNN，每个网络都是 24 个 softmax 输出。

1. 三层全连接网络

为了简单起见，实验用三层全连接网络，如图 6-3 所示。

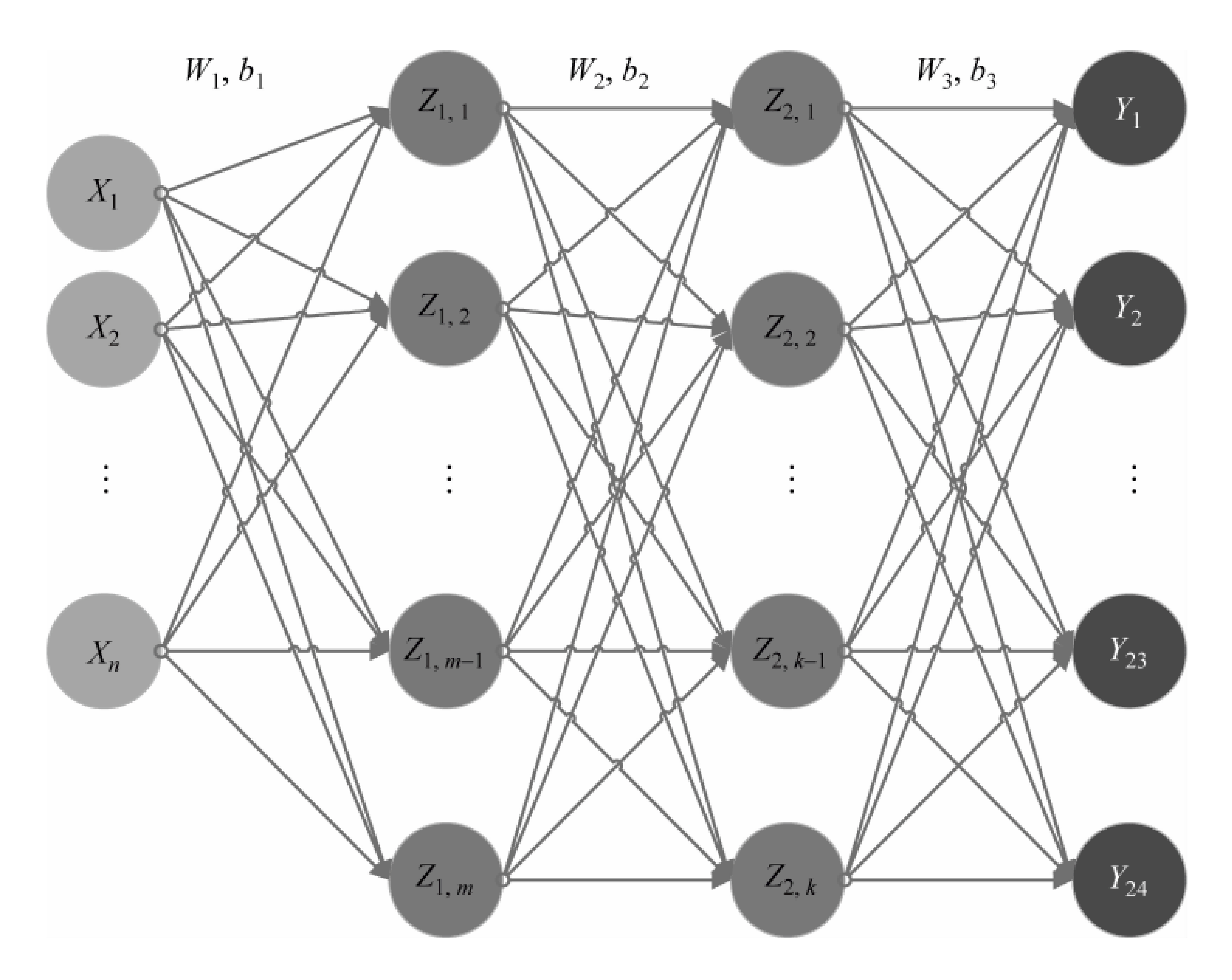


图 6-3　实验用的三层全连接网络

该网络采用最简单的多层全连接的网络模式，一共有三层。每一层（设为第 i 层）隐含层（也包括输入层）的输入 Z_i 会经过一个线性环节 W_i 和 b_i，得到输出 W_Z+b，然后将输出输入到一个 ReLU 环节进行运算，得到这层神经网络的真实输出，表示为 O_i。其中 $Z_{i+1}=O_i$。在输出层，会进行类似的处理，不过最终的输出会进入一个 softmax 环节，将计算所得的结果归一化。

2. CLDNN

实验用的语音识别网络结构参考谷歌公司的 CLDNN 结构。CLDNN 由三部分组成：卷积层、长短时记忆网络（Long Short-Term Memory，LSTM）和全连接层。

实验采用类似于 CLDNN 的 8 层网络，包含一个傅里叶变换层，加上 4 个卷积网络，再加上双向 LSTM 层，最后是两个全连接层。整体结构如图 6-4 所示。

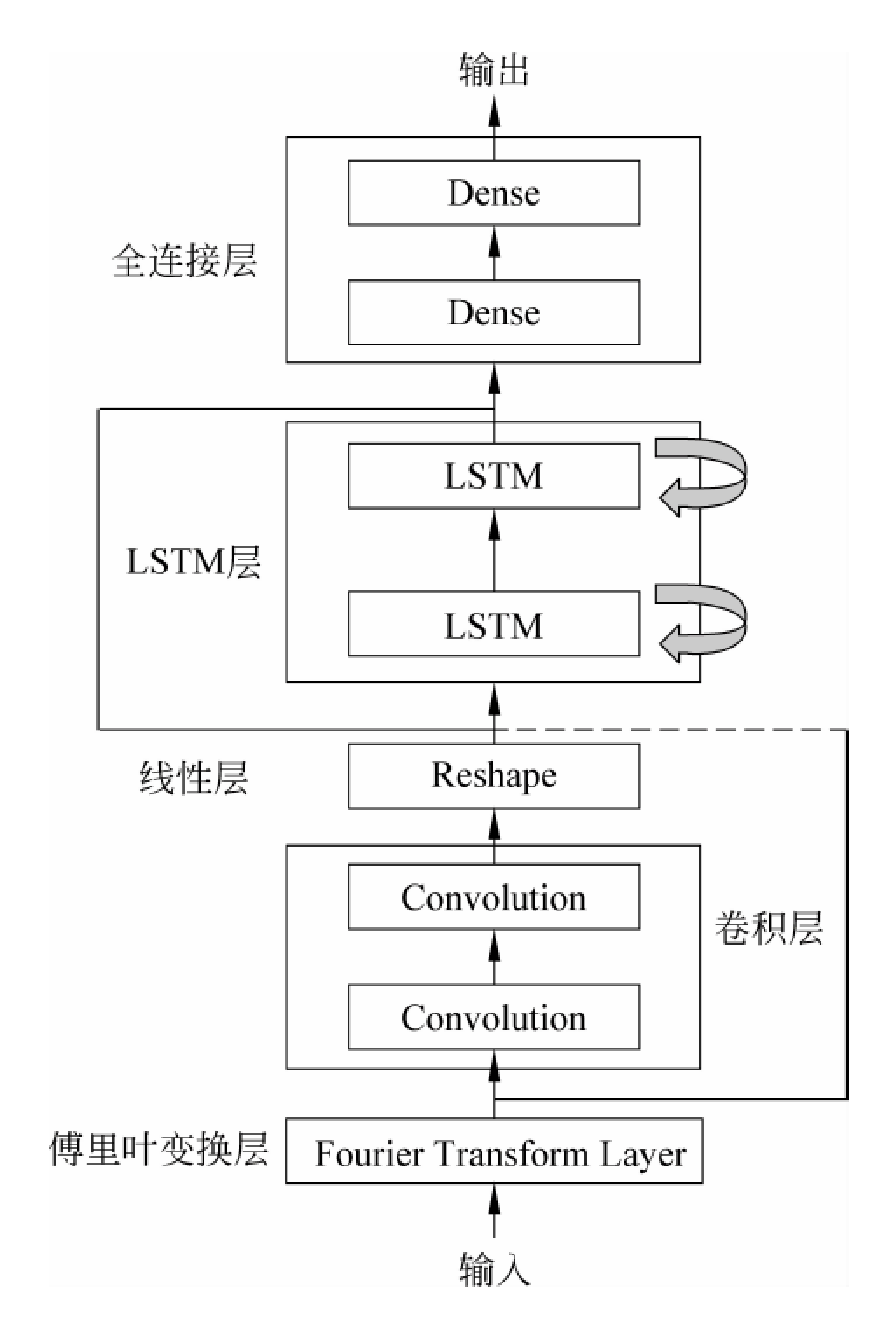


图 6-4　实验用的 8 层 CLDNN

3. 交叉熵损失函数

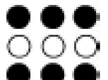
在训练过程中，最终网络的输出结果会与实际所对应的标签进行对比。具体的方式是通过一个交叉熵的公式，计算损失函数。神经网络的训练过程就是损失函数的最小化过程。

交叉熵形式的损失函数表示如下：

$$H_{y'}(y)=-\sum_{i} y'_i \ln y_i$$

其中，y'表示训练样本对应的标签，y表示神经网络的输出。

实际中还会在损失函数中加上一些正则项(regularizer)或惩罚项(penalty term)，该过程称为正则化(regularization)。目的是为了防止过学习(over-fitting)。常见的惩罚项是$\alpha\sum_{j}\sum_{i}\|W_{ij}\|$。



6.2.3 TensorFlow/Keras 的使用

为了便于上手,代码框架采用 TensorFlow/Keras 框架。

用 Keras 定义的语音识别网络结构,定义在 Keras 的 Model 中,如下所示。

```
defKerasModel(isCompile=True):
    In =Input(shape=(None, 1, 1))
    x =Lambda(fourierLayer, output_shape=fourierLayerShape)(In)
    x =Convolution2D(32, 7, 7, subsample=(3,3), activation='relu',
        W_regularizer=l2(L2_RATE))(x)
    #x =Dropout(DROP_RATE)(x)
    x =Convolution2D(64, 7, 5, subsample=(3,3), activation='relu',
        W_regularizer=l2(L2_RATE))(x)
    #x =Dropout(DROP_RATE)(x)
    x =Convolution2D(64, 3, 3, subsample=(2,2), activation='relu',
        W_regularizer=l2(L2_RATE))(x)
    #x =Dropout(DROP_RATE)(x)
    x =Convolution2D(32, 3, 3, subsample=(2,2), activation='relu',
        W_regularizer=l2(L2_RATE))(x)
    #x =Dropout(DROP_RATE)(x)

freq, chan =x.get_shape()[2:4]
    x =TimeDistributed(Reshape([int(freq) * int(chan)]))(x)
    x =Bidirectional(LSTM(128, ))(x)
    #x =Dropout(DROP_RATE)(x)
    x =Dense(64, activation='relu')(x)
    #x =Dropout(DROP_RATE)(x)
    x =Dense(24, activation='softmax')(x)
model =Model(input=In, output=x)
```

1. GPU 工作站准备

实验采用微软 Azure 云提供的带 GPU 的 Windows Server 2012 虚拟机。该 GPU 服务器虚拟机上已经安装了 Python 3.5 和 TensorFlow 1.0 版本。在 GPU 虚拟机中使用 Python 前,需要事先激活 Python 3.5 环境。

如果有带 GPU 卡的 Windows Server 2012 工作站,可以下载 CUDA Toolkit。CUDA Toolkit 是 NVIDIA 公司提供的 GPU 编程基础工具包,目前版本是 8.0,下载地

址如下：

```
https://developer.nvidia.com/cuda-downloads
```

在神经网络训练过程中，为了加速深度学习过程，安装 CUDNN 库。CUDNN 库的目前版本是 6.0，下载地址为

```
https://developer.nvidia.com/cudnn
```

解压出来是名为 CUDA 的文件夹，里面有 bin、include 和 lib 3 个目录。将这 3 个目录中的文件夹复制到 CUDA 的安装路径下，覆盖对应的 bin、include 和 lib 文件夹。CUDA 默认的安装路径为

```
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\8.0
```

2. 数据增强

在神经网络训练过程中，为了解决训练样本不足和网络的抗噪性问题，我们采用数据增强(Data Augmentation)方法。数据增强方法是在原始训练数据上加上高斯白噪声等扰动信号，增加可供训练的样本数目。

实验中，SOX 程序(SOund eXchange)用于对原始语音录音文件，进行添加噪声、混响等扰动信号，增加可供训练的样本数目。

SOX 程序的下载地址为

```
http://sox.sourceforge.net/
```

3. 实验过程

具体实验步骤和过程如下。

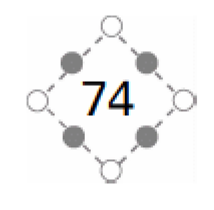
1) 下载 audioNet 项目代码

用 git 或者直接在网页下载压缩包。下载地址为

```
http://gitlab.icenter.tsinghua.edu.cn/saturnlab/audioNet
```

2) 下载训练数据

用于训练的数据是经过 ffmpeg 格式统一后的音频文件，形如 10000_01.wav，以压缩包的形式给出。



3）创建 models 目录

假设 audioNet 代码下载解压后的目录是/audioNet/，创建 models 目录，用于保存训练过程中暂存的模型文件。

```
$ mkdir ./audioNet/models
```

4）制定训练数据的目录

修改/audioNet/train.py，将其中涉及目录路径的地方，改成对应平台的目录。例如在 Windows 下，目录为“.\\ models\\”；在 Linux 下目录为“./models/”。

修改/audioNet/augmentation/client.py 的最后几行，将其涉及数据文件目录的地方改成相应的位置。例如，如果数据放在 D 盘，就得改成类似于“D:\\data\\ *.wav”。

5）运行训练与测试

运行代码 train.py：

```
$ python train.py
```

运行后会打开一个开启训练的进程，它在等待数据。

新建一个命令行窗口，运行 client.py：

```
$ python augmentation/client.py
```

运行后会新建几个进程，读取 client.py 末尾提供的 train 的目录文件，然后传给 train.py 的进程训练，读取 client.py 末尾提供的 test 的目录文件，然后进行测试。训练的效果会实时显示。

训练的迭代轮次可以在 train.py 中修改，默认是 20 轮保存一次模型文件。保存模型文件的代码如下：

```
model.save_weights('.' +os.sep +'models' +os.sep +'save_' +str(i) +'.h5' )
```

运行时，会在 models 文件夹下生成一系列模型 save_xx.h5 文件，如图 6-5 所示。

save_01.h5　2017/6/
save_02.h5　2017/6/
save_03.h5　2017/6/
save_04.h5　2017/6/

图 6-5　网络模型文件

训练完毕后，选取 models 文件夹的最新的 save_xx. h5 文件。

6）推断过程

推断过程在 predict. py 文件中，首先新建一个 KerasModel 类，接着调用 load_weights 函数，加载训练过程中保存的模型文件，然后调用 modelA. predict( ) 函数，给出推断的结果。

```
modelA =KerasModel(False)
modelA.load_weights(modelfile)
 ⋮
plb =modelA.predict(wav, 1)
```

7）运行网页端

修改/audioNet/webfront. py 的第 86 行，将 model 文件路径设定为当前生成的 model 文件中的一个。之后，也会使用该模型文件作为推断模型。代码如下：

```
os.system(ffmpeg_cmd)
res =predict(fullpath +'.wav', './models/save_14.h5')
else:
flash('Unable to find ffmpeg, please install ffmpeg')
return redirect(request.url)
```

运行命令：

```
$ python webfront.py
```

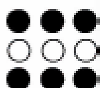
运行后将打开一个网页服务，可以提供在线语音识别服务。

6.3 小结

本章介绍了一个语音控制功能的实现，主要有训练数据预处理过程和深度神经网络的训练过程两大部分。

训练数据的预处理过程包括语音指令的设计、录音文件采集、音频文件格式转换与时频谱图生成。

深度神经网络的训练过程包括网络结构设计、损失函数设定、TensorFlow/Keras 的实现和数据增强等技术。



实验使用 3 层 FCN 和 8 层 CLDNN 做对比测试，验证网络架构对识别的效果影响。普遍来说，采用数据增强技术将使得网络的鲁棒性和抗噪性更好。

参考文献

[1] Spectrum[EB/OL]. http://gitlab.icenter.tsinghua.edu.cn/saturnlab/spectrum.

[2] asrNet[EB/OL]. http://gitlab.icenter.tsinghua.edu.cn/saturnlab/asrNet.

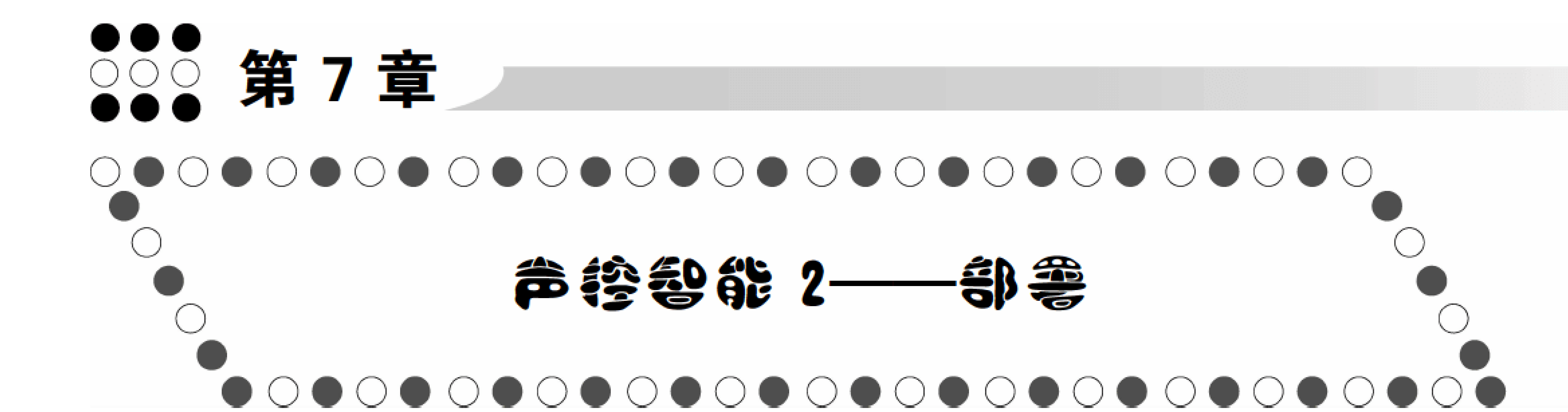
[3] audioNet[EB/OL]. http://gitlab.icenter.tsinghua.edu.cn/saturnlab/audioNet.

[4] Keras document[EB/OL]. http://keras.io.

[5] T N Sainath, O Vinyals, A Senior, et al., Convolutional, Long Short-Term Memory, Fully Connected Deep Neural Networks[C]// IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE, 2015:4580-4584.

[6] audioPlot[EB/OL]. http://gitlab.icenter.tsinghua.edu.cn/saturnlab/audioPlot.

[7] T N Sainath, et al. Multichannel Signal Processing With Deep Neural Networks for Automatic Speech Recognition[J], IEEE TASLP, 2017,25(5).



第 7 章 声控智能 2——部署

深度学习最重要的功能是进行推断(Inference)。我们可以把推断功能放在网站端、云端或服务器端(称为在线推断),也可以把推断功能放在移动端(又称为离线推断)。

在线推断的优势是可以随时更新神经网络的参数,用户不需要下载模型文件,只需要调用 Web API 即可完成。

离线推断的优势是模型文件直接部署在终端设备,没有网络通信的延迟,适合网络通信不畅通或者没有网络的情况。

7.1 网站端——在线推断

7.1.1 云知音网站功能

移动终端(Client)只需要打开浏览器,利用浏览器完成如下功能。

(1) 调用浏览器 API 录音。

(2) 发送录音文件。

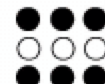
(3) 接收云端(Server)的反馈并显示。

云知音网站(Server)的功能如下。

(1) 接收录音文件。

(2) 调用服务端程序。

(3) 运行 Keras/TensorFlow,对语音文件进行推断。



（4）返回结果。

由于指定的功能相对简单，为了便于开发，我们采用 Flask 框架架构语音识别功能。

7.1.2 Flask 网站搭建

搭建基于 Flask 的网站，准备 virtualenv。安装方法（下面两条命令适用于 Mac 和 Linux）如下：

```
$ sudo easy_install virtualenv
```

或者是：

```
$ sudo pip install virtualenv
```

如果使用 Ubuntu，请尝试：

```
$ sudo apt-get install python-virtualenv
```

如果使用 CentOS，请尝试：

```
$ sudo yum install python-virtualenv
```

安装好 virtualenv 后，可以创建一个项目文件夹，利用 virtualenv 命令在其下创建 venv 文件夹：

```
$ mkdir myproject
$ cd myproject
$ virtualenv venv
New python executable invenv/bin/python
Installing distribute…done.
```

现在，可以根据项目的具体工作环境，激活相应的环境。在 Linux 下，按如下做：

```
$  .venv/bin/activate
```

现在，只需要输入以下命令来激活 virtualenv 中的 Flask：

```
$ pip install Flask
```

7.1.3 Flask＋Keras 实现

webfront.py 是一个示例性的 Flask 程序，用于演示语音在线推断功能。

首先，定义字典表 DICT。

```
DICT=[
    '蓝牙开机',     '蓝牙拨打电话',     '蓝牙打电话',     '蓝牙接听电话',
    '蓝牙接电话',     '蓝牙拒接',     '蓝牙播放音乐',     '蓝牙开始音乐',
    '蓝牙暂停音乐',     '蓝牙停止音乐',     '蓝牙上一首',     '蓝牙上一曲',
    '蓝牙下一首',     '蓝牙下一曲',     '蓝牙音量增大',     '蓝牙声音增大',
    '蓝牙音量增加',     '蓝牙声音增加',     '蓝牙音量减小',     '蓝牙声音减小',
    '蓝牙关机',     '蓝牙电量提醒',     '蓝牙还剩多少电',     '蓝牙还剩多少电量'
]
```

```
import os, sys, subprocess
import numpy

from flask import Flask, request, redirect, flash
from wavReader import readWav
from model import KerasModel

UPLOAD_FOLDER = '/tmp/audioNet'

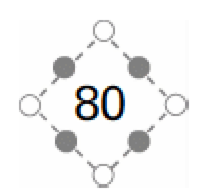
app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

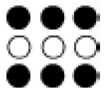
def predict(wavfile, modelfile):
    spl, wav = readWav(wavfile)
    wav = wav.reshape([1, -1, 1, 1])

    modelA = KerasModel()
    modelA.load_weights(modelfile)

    result = modelA.predict(wav, 1)
    return numpy.argmax(result, 1)

@ app.route('/predict', methods=['GET', 'POST'])
```





```
def predictAction():
    if request.method =='POST':
        #check if the post request has the file part
        ⋮
            if command_exists('ffmpeg'):
                ffmpeg_cmd ='ffmpeg -i {} -ac 1 -acodec pcm_f32le -ar 44100 {}.
wav -v 1'.format(fullpath, fullpath)
                os.system(ffmpeg_cmd)
                res =predict(fullpath +'.wav', './models/save_14.h5')
            else:
                flash('Unable to find ffmpeg, please install ffmpeg')
                return redirect(request.url)

            return str(DICT[res])

    return '''
<! doctype html>
<title>audioNet Predict</title>
<h1>Upload audio for predict</h1>
<form action="" method=postenctype=multipart/form-data>
<p><input type=file name=file accept='audio/* ' capture>
<input type=submit value=Upload>
</form>
    '''


    if __name__ =="__main__" :
        if not os.path.exists(app.config['UPLOAD_FOLDER']):
            os.makedirs(app.config['UPLOAD_FOLDER'])
        app.run(host='0.0.0.0', port=5000)
```

7.2 移动端——离线推断

7.2.1 移动端的网络模型文件

1. 网络模型文件转换

TensorFlow 作为在 GPU 工作站或高性能服务器上使用的深度学习框架，通过在数

据集上训练生成的深度神经网络模型，可以转换为在移动设备上能够使用的网络模型文件。

为了能够在安卓移动终端使用 TensorFlow 训练的模型文件，TensorFlow 提供了一套工具库，方便将网络模型文件从用于训练的工作站上导出。

同时，TensorFlow 提供了模型的优化工具，降低空间消耗，便于在移动平台上使用。

网络模型文件的导出过程，分为 Graph_def 文件生成、model. ckpt 文件生成、冻图和优化等几个环节。

2. 生成 model. pb 文件

使用 TensorFlow 创建好网络模型，给变量赋值后，使用代码生成 gDef 对象：

```
from tensorflow.python.framework import

graph =tf.get_default_graph()
gDef =graph.as_graph_def()
```

gDef 是一个 protobuf 对象，可以通过两种方式将其转化为字节序列：

```
str(gDef)：将其转化为可读的字节序列。
gDef.SerializeToString()：将其转化为不可读的二进制编码，节省空间。
```

安卓移动终端需要的是不可读的二进制编码形式，所以采用 SerializeToString()方法，将其保存到 xxx_graph_def. pb 文件中：

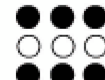
```
with open("model.pb", "wb") as f:
    f.write(gDef.SerializeToString())
```

注意：在一个 graph 中，权重是以变量(variable)的形式提供的，gDef 对象不会保存权重的数值。另外，在 gDef 对象中，通常会包含很多我们不想要的节点。

3. 生成 model. ckpt 文件

通过 TensorFlow 框架生成的网络模型，可以很方便地用 saver 类将模型文件保存为 model. ckpt-xxx 文件。

Keras 是基于 TensorFlow 框架的。如果训练时使用 Keras 来生成网络模型，则需要新建一个 KerasModel 类，接着调用 load_weights 函数，加载训练过程中保存的模型文件。再使用 TensorFlow 的 saver 类，将模型文件保存为 model. ckpt-xxx 文件。这样就



完成了 Keras 模型文件到 TensorFlow 模型文件的转换。

从 Keras 模型文件到 TensorFlow 模型文件的转换过程大致用代码描述如下：

```
import tensorflow as tf
from model import KerasModel
from tensorflow.python.training import saver
from tensorflow.python.framework import graph_io
from keras import backend as K

⋮

modelA =KerasModel(False)
modelA.load_weights(modelfile)
⋮
init_all_op =tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init_op)
    saver_path =saver.save(sess, "save/modelA.ckpt")
print("Model saved in file:", saver_path)
```

采用 Keras 调用后端 TensorFlow 的 session，将 Keras 模型转换为 TensorFlow 模型文件。

```
from model import KerasModel
from tensorflow.python.training import saver
from tensorflow.python.framework import graph_io
from keras import backend as K

⋮

modelB =KerasModel(False)

modelB.load_weights(modelfile)

⋮

sess =K.get_session()
```

```
ckpt_path = saver.Saver().save(sess, "models/modelB.ckpt")

graph_io.write_graph(sess.graph, './models', 'modelB.pb')
```

4. 冻图和优化

这时，就有了两种形式的文件，即 *.pb 和 *.ckpt 文件。前面提到，在导出的 graph 的 *.pb 文件中，权重是以变量(variable)的形式提供的，不会被保存。而且，graph 通常会包含很多我们不想要的节点，浪费了很多空间。

在 TensorFlow 的 python/tools 目录中，TensorFlow 提供了 freeze_graph.py 和 optimize_for_inference.py 工具，可以解决上述两个问题。

```
tensorflow/python/tensorflow/tools/
```

其中，freeze_graph.py 将变量转化为常数(constant)，解决变量的问题；而 optimize_for_inference.py 用来删除无用节点。具体工具的用法，参考同目录下对应的 xxx_test.py 就可以了。

freeze_graph.py 命令如下：

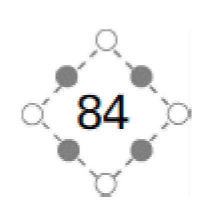
```
$ python -m tensorflow.python.tools.freeze_graph --input_graph=./models/modelA.pb --input_checkpoint=./models/modelA.ckpt --output_graph=./asrModel.pb --output_node_names="dense_2/Softmax"
```

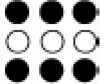
freeze_graph.py 的使用说明如下：

```
freeze_graph \
--input_graph=xxx_graph_def.pb \
--input_checkpoint=xxx_model.ckpt \
--output_graph=xxx_graph.pb \
--output_node_names=softmax \
⋮
```

optimize_for_inference.py 的使用说明如下：

```
optimize_for_inference \
--input=frozen_xxx_graph.pb \
--output=optimized_xxx_graph.pb \
--frozen_graph=True \
⋮
```





工具目录下还有许多工具，用法参考同目录下对应的 xxx_test. py。

TensorFlow 的 pip 包一般不包含这些工具。这些工具可以直接手动复制过来使用，但是要考虑一些依赖关系。

其中，optimize_for_inference. py 依赖于相同目录下的 strip_xxx. py 的文件，所以需要复制如下 3 个文件过来。

```
strip_unused.py
strip_unused_lib.py
strip_unused_test.py
```

同时，修改 optimize_for_inference. py 的 import 方式，将

```
import tensorflow.python.tools.strip_xxx
```

改为

```
import strip_xxx
```

freeze_graph. py 没有需要额外处理的依赖。

直接使用经过 freeze_graph. py 冻图后的模型文件，而没有进一步使用 optimize_for_inference. py 来做优化处理。

另外，在该例子中，optimize_for_inference. py 有点问题，处理出来的 graph 会导致 session 的创建失败。

7.2.2 安卓平台的 TensorFlow 库生成

安卓平台的硬件多采用 ARM 处理器，并且安卓应用一般采用 Java 语言开发。而 TensorFlow 的核心是用 C++ 写的。为了让 TensorFlow 代码能在安卓平台上使用，需要编译出相应的类库文件，即 libtensorflow_inference. so 和 libtensorflow-core. a。

1. libtensorflow_inference. so

安卓应用是采用 Java 语言开发的，TensorFlow 框架的安卓 Java 库提供 Java 的接口，称为 TensorFlowInferenceInterface 的类。该类是以 JNI(Java Native Invocation)的形式提供。

首先，使用 Linux 下的 Android Studio 作为开发环境。使用 Android Studio 创建一

个带 C++ 支持的新项目。

打开项目后，在 Android Studio 的 SDK Manager 中，安装 NDK 和一些 Build Tool 的工具。

具体需要安装哪些工具，在 Build 过程的报错信息中会有提示。其中，Build Tool 对版本有要求，需要在 SDK Manager 界面点开选项，来选择指定版本。

下载 tensorflow-master 的压缩包，或者用 git clone 下载某个版本的 TensorFlow，将其根目录置于 Android 项目之下。完成之后的目录结构形如：

```
'myapp/tensorflow-master/tensorflow/contrib/...'
```

找到 README.md 文件：

```
myapp/tensorflow-master/tensorflow/contrib/android/cmake/README.md
```

根据 README.md 的说明，整理的 Android-Inference 的操作如下。

(1) 在 Android Studio 中，找到整个 Project 的 setting.gradle (位于 myapp/setting.gradle)，在末尾添加：

```
include ':TensorFlow-Android-Inference'
findProject(":TensorFlow-Android-Inference").projectDir =
new File("${/path/to/tensorflow_repo}/contrib/android/cmake")
```

注意：其中 ${/path/to/tensorflow_repo} 需要自己填写，最好填写相对目录。例如，本例子中就填写为 ./tensorflow-master/tensorflow/contrib/android/cmake/。

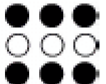
(2) 新建项目有且只有一个子模块，一般称为 app，找到其所属的 build.gradle。

该文件最末尾有一个由花括号圈起来的 dependencies 块，在其中添加两行：

```
debugCompile project(path: ':TensorFlow-Android-Inference', configuration:
'debug')
releaseCompile project(path: ':TensorFlow-Android-Inference', configuration:
'release')
```

注意：path 的内容和原网页不同。

```
TensorFlow-Inference-debug.aar
TensorFlow-Inference-release.aar
```



libtensorflow_inference. so 文件打包了以上 * . aar 文件,并放置于以下目录:

```
jni/$ {ANDROID_ABI}/
```

(3) 重启 syncgradle,或者重启 Android Studio。目的是重新扫描 gradle 的配置文件。

此后,应该可以看到 Android Studio 找到了一个新的子模块:

```
TensorFlow-Android-Inference
```

(4) 修改 TensorFlow-Android-Inference 所属的 build. gradle。

2. libtensorflow-core. a

现在的目标是新建过程编译出 TensorFlow 的核心库 libtensorflow-core. a。由于编译过程很长,所以 build. gradle 默认是不编译的。现在确实要编译核心库,我们仅需要根据 build. gradle 注释,找到被//注释的一行,删除//就可以了。

同时,从注释中也可以看出,这个操作只对 Linux 和 Mac OS 有效,在 Windows 下需要自己想办法编译出 libtensorflow-core. a,然后放到相应的文件目录下:

```
tensorflow/contrib/makefile/gen/lib/libtensorflow-core.a
```

如果在安卓移动终端上使用这个 * . a 文件,因为安卓移动终端多采用 ARM 硬件,需要交叉编译出针对 ARM 平台的 TensorFlow 代码。所以,建议采用 Linux 工作环境。

(1) 添加可用平台列表。

打开 TensorFlow-Android-Inference 的 build. gradle,可以发现一行:

```
ndk{
    abiFilters "armeabi-v7a"
}
```

这说明 TensorFlow 的安卓版只支持 armeabi-v7a 这个平台。

ABI(Application Binary Interface)是指应用程序将基于哪种指令集进行编译。ABI 可供选择的编译选项只有 5 种:armeabi、armeabi-v7a、arm64-v8a、x86 和 mips 。其中,armeabi 和 armeabi-v7a 为最常见的。

armeabi 是指创建以基于 ARM * v5TE 的设备为目标的库。具有这种编译目标的浮点运算使用软件浮点运算。使用 armeabi 创建的二进制代码,将可以在任何 ARM 设

备上运行。

armeabi-v7a 是指创建支持基于 ARM v7 设备的库，具有这种编译目标的浮点运算将使用硬件 FPU 浮点运算指令。这是默认选项。

arm64-v8a 是指 64 位 ARM 设备。

在项目中的 build. gradle 添加这一行。build. gradle 没有 NDK 这个项，需要自己写，位置是在 defaultConfig 下边。

(2) 在 Android Studio 中进行创建，生成输出文件 libtensorflow-core. a。

更详细的过程，请参考 TensorFlow 主页的 github 网址链接：

```
https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/
android/cmake
```

7.2.3 安卓应用的 TensorFlow 库调用

1. TensorFlowInferenceInterface

安卓应用调用 TensorFlow 库的一个最简单的方法，就是在新建项目默认的 Activity 类中，添加一个成员变量，类型为 TensorFlowInferenceInterface。

首先，声明使用 TensorFlowInferenceInterface 类型。

```
import org.tensorflow.contrib.android.TensorFlowInferenceInterface
```

在 AndroidStudioIDE 中，使用快捷键 Alt + Enter，自动添加 import TensorFlowInferenceInterface。

添加一个成员变量，类型为 TensorFlowInferenceInterface，名为 tfii：

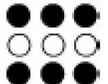
```
private TensorFlowInferenceInterface tfii;
```

然后在 onCreate 函数中，对 tfii 进行初始化。

```
tfii.initializeTensorFlow(getAssets(), modelFilename)
```

其中：

getAssets() 方法是 Activity 的成员函数，用于获得 AssetsManager 实例。
modelFilename 是模型文件的路径，其形式为 "file:///android_asset/xxx.x"。



由于安卓应用跑在沙盒虚拟机(Sandbox)中,无法访问绝对路径。所以,以上文件名会被映射到安装目录的路径中,以做到权限控制。

2. 载入推断模型文件

在 Android Studio 中新建一个文件夹,取名为 assets。

然后,将 7.2.2 节冻图后的网络模型文件 xxx.pb 放入这里,通过文件目录 file:///android_asset/xxx.x 访问。由 TensorFlowInferenceInterface 自己负责访问该模型文件。

在这里,这个函数的返回值只有 0 代表成功,否则都是失败,并且经常不会通过返回值告诉失败,而是直接闪退。

具体的使用,可以参考:

```
./tensorflow/examples/android/src/org/tensorflow/demo/
TensorFlowImageClassifier.java
```

7.2.4 安卓应用的录音功能调用

1. 录音权限申请

在安卓应用中调用手机的录音(Record)功能,首先需要获得系统的录音权限,否则会出现各种错误提示。

权限获取操作很简单,需要修改 app 项目中的 manifest.xml 文件。manifest.xml 作为一个 xml 文件,其唯一根节点名为 manifest,并且包含一系列属性。其下,包含 Activity 节点和其他节点。

添加权限使用的节点与 Activity 同级,名为 uses-permission。添加录音权限的声明,其完整形式为

```
<uses-permissionandroid:name="android.permission.RECORD_AUDIO" />
```

添加完权限后,最好重新编译(Rebuild)一遍,确保生效。

在某些手机上,系统会在安卓体系之外建立权限管理,所以很有可能应用虽然申请了权限,但是仍然需要用户在手机界面点确认,或者在权限管理里面手动打开权限开关。

2. AudioRecord 方法

使用 AudioRecord 类进行录音,好处是录音结果就是内存中的数组,以浮点数、整数

的数组形式给出，方便算法处理。

该类的运行模式很简单：申请一块缓存，每当开始录音，就把数据存入缓存。AudioRecord 不断向缓存中存数据，维持运行链条。如果缓存满了，就会报 over running 错误。

在使用上，首先要确定录音的几个参数。

(1) 从哪里录？（只有一个选择，MediaRecorder. AudioSource. MIC）

(2) 采样率是多少？（比较常见的是 44 100）

(3) 几个通道？（单声道 AudioFormat. CHANNEL_IN_MONO，或者双声道 AudioFormat. CHANNEL_IN_STEREO）

(4) 每个样本用几位存储？（必须用 PCM 格式，同时可以选择 8b 整数、16b 整数或者 32b 浮点数，AudioFormat. ENCODING_PCM_8BIT/16BIT/FLOAT）

确定了以上这几个参数，就可以调用 AudioRecord. getMinBufferSize()来获得维持录音所需的最小缓存空间。

将上述参数，以及缓存大小输入给 AudioRecord 的构造函数，就可以创建一个 AudioRecord 类的实例，例如名为 ar。

调用 ar. startRecording()就可以启动录音。如果什么都不做，缓存马上就会被填满，然后报错。

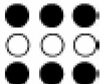
调用 ar. stop()可以停止录音。然后再调用 ar. release()可以释放录音所占用的资源（例如麦克风的所有权）。

调用 ar. read()可以从缓存中取数据，从而清理缓存。

为了维持连续运行，需要通过多线程+循环来实现录音，例如：

```
buffer =short[100] // 100 >AudioRecord.getMinBufferSize()
ar.startRecording()
while(isRecording){
    L =ar.read(buffer, pos, 100)
    pos +=L
}
ar.stop()
ar.release()
```

上述是线程函数，isRecording 是控制线程运行的变量。在线程外，改变 isRecording



为 false，就能停止录音。

7.2.5 快速集成开发

快速集成开发是将以上内容实例化，快速完成语音识别 APP 的制作过程。实验环境如下。

（1）Android Studio 2.3 版本及以上。

（2）华为 MediaPad 2 平板，操作系统为 Android 6.0，CPU 型号麒麟 930。

具体实验步骤和过程如下。

（1）下载 androidAudioRecg 项目代码：

```
http://gitlab.icenter.tsinghua.edu.cn/saturnlab/androidAudioRecg
```

用 git 或者直接在网页下载压缩包。

（2）下载 libtensorflow 的 jar 文件和 libtensorflow_inference.so。

获取 TensorFlow 的依赖包 libtensorflow 的 jar 文件和 libtensorflow_inference.so 的系列文件压缩包，如图 7-1 所示。



图 7-1 获取 libtensorflow 库文件压缩包

下载地址为

```
https://ci.tensorflow.org/view/Nightly/job/nightly-android/
```

(3) 将 jar 文件与压缩包内容放到 androidAudioRecg 项目的 app/libs 下，如图 7-2 所示。



图 7-2　放置 jar 文件与压缩包内容

如果 TensorFlow 版本为 1.2 以上，TensorFlow 推断接口(Inference Interface)在 JCenter 中可以直接调用，只需在 build.gradle 文件中写入，Gradle 会自动使用最新 tensorflow.aar 文件。

(4) 安装 Android Studio，然后使用 SDK Manager 安装较新的 SDK，如图 7-3 所示。

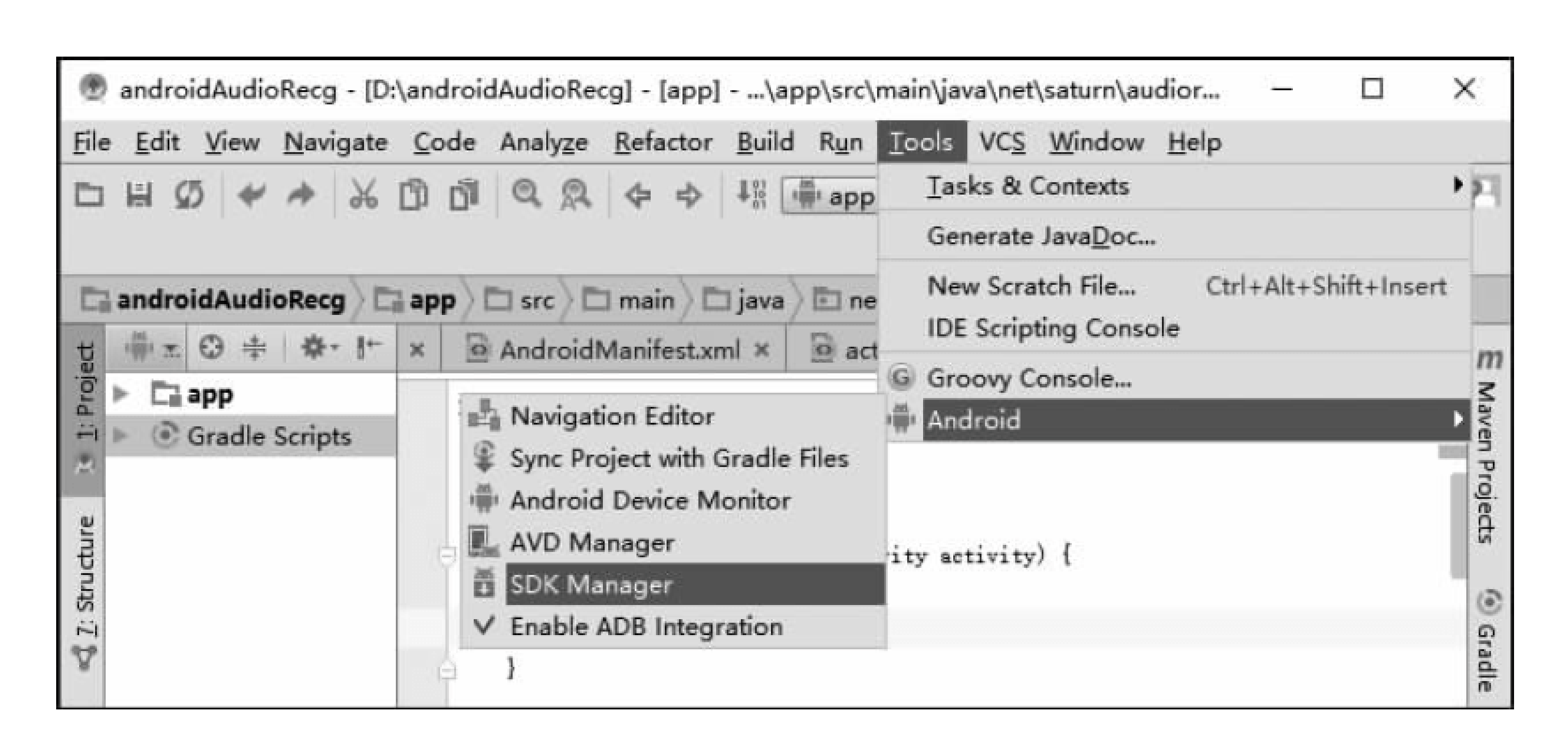
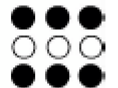


图 7-3　安装较新的 SDK

我们需要 Android 6.0 以上版本，选择 Android 6.0 及以上，更新 SDK 即可，如图 7-4 所示。

(5) 用 Android Studio 打开项目。



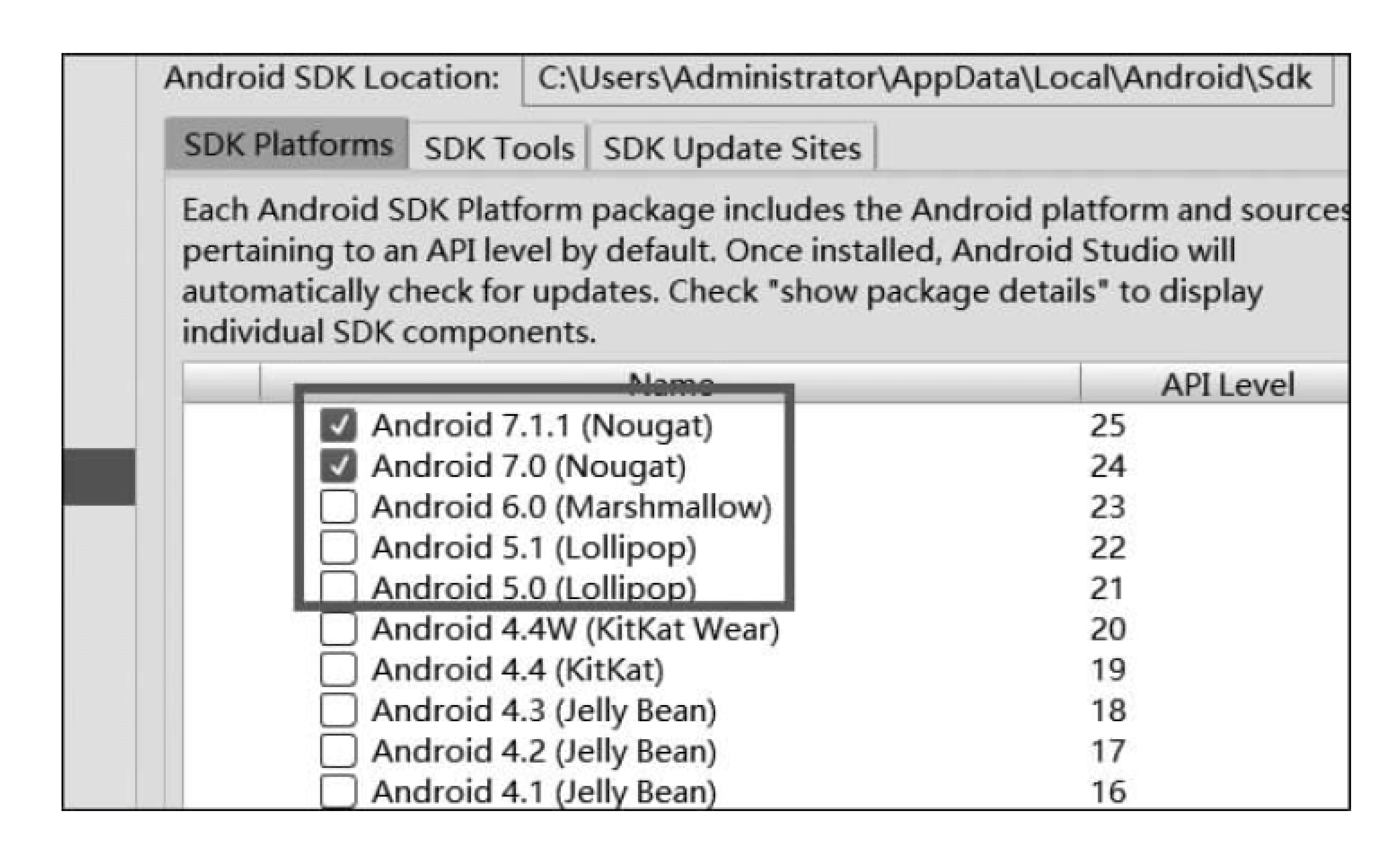


图 7-4　选择 Android 6.0 以上版本

① 修改 manifest，使 APP 具有录音权限。

② 编辑 APP 应用界面的布局。

③ 编辑录音按钮的属性。

(6) 插上手机，打开手机的开发者模式，如果是华为手机或平板，操作如下。

① 进入手机的设置，在其版本号的位置，连击 4 次，进入开发者模式。

② 进入开发者模式，选择打开 USB 调试功能。

(7) 用数据线连接手机到计算机，在 Android Studio 中让 APP 在手机上运行调试应用。

7.3 小结

本章介绍将训练好的神经网络模型文件，分别部署在网站端和移动平板端的示例。网站端采用 Flask 框架，调用 Keras 模型来完成推断。而在移动平台端，需要事先将编译的库文件放入指定目录。

网站端在线部署的优势是可以不断采集用户的数据，扩大语音识别的训练集数据。同时，可以随时更新神经网络参数。用户不需要下载任何文件，只需要打开网站，调用 Web API 即可完成。

移动端离线部署的优势是可以离线使用，网络模型直接部署在终端设备上，没有网

络通信的延迟,不依赖网络通信。

参考文献

[1] Miguel Grinberg. Flask Web Development: Developing Web Applications with Python[M]. Sebastopol: O'Reilly Media, Inc., 2014.

[2] audioNet[EB/OL]. http://gitlab.icenter.tsinghua.edu.cn/saturnlab/audioNet.

[3] Android Studio[EB/OL]. https://developer.android.com/studio/index.html.

[4] TensorFlow[EB/OL]. http://www.tensorflow.org.

[5] Android TensorFlow Example[EB/OL]. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android.

[6] Android Audio Recognition[EB/OL]. http://gitlab.icenter.tsinghua.edu.cn/saturnlab/androidAudioRecg.

[7] TensorFlow Android[EB/OL]. https://ci.tensorflow.org/view/Nightly/job/nightly-android/.

第 8 章



在第 7 章，已经初步完成了基于 TensorFlow 的语音识别功能。这里我们设计实验，通过 PYNQ 开发板进行 Audio 录音、录音文件转换、发送录音文件和接收服务器反馈。

8.1 PYNQ

8.1.1 PYNQ 简介

PYNQ(Python for Zynq)是 Xilinx 公司生产的一款嵌入式智能硬件，如图 8-1 所示。PYNQ-Z1 开发板支持 PYNQ 项目，这是一个新的开源框架，使嵌入式编程人员能够在无须设计可编程逻辑电路的情况下即可充分发挥 Xilinx Zynq All Programmable SoC (APSoC)的功能。

通过 PYNQ，用户可以使用 Python 进行 APSoC 编程，包括对 FPGA 编程，并且代码可直接在 PYNQ-Z1 上进行开发和测试。

PYNQ 外接传感器模块 Pmod，不同的 Pmod 具有不同的传感器功能，如光亮度、超声波、温度和湿度测量等。PYNQ 可以通过 Python 直接采集传感器的数值，并进行反应控制。

8.1.2 PYNQ-Z1 开发板

首先，需要了解安装 PYNQ 板的软硬件环境。ZYNQ XC7Z020-1CLG400C 是 PYNQ

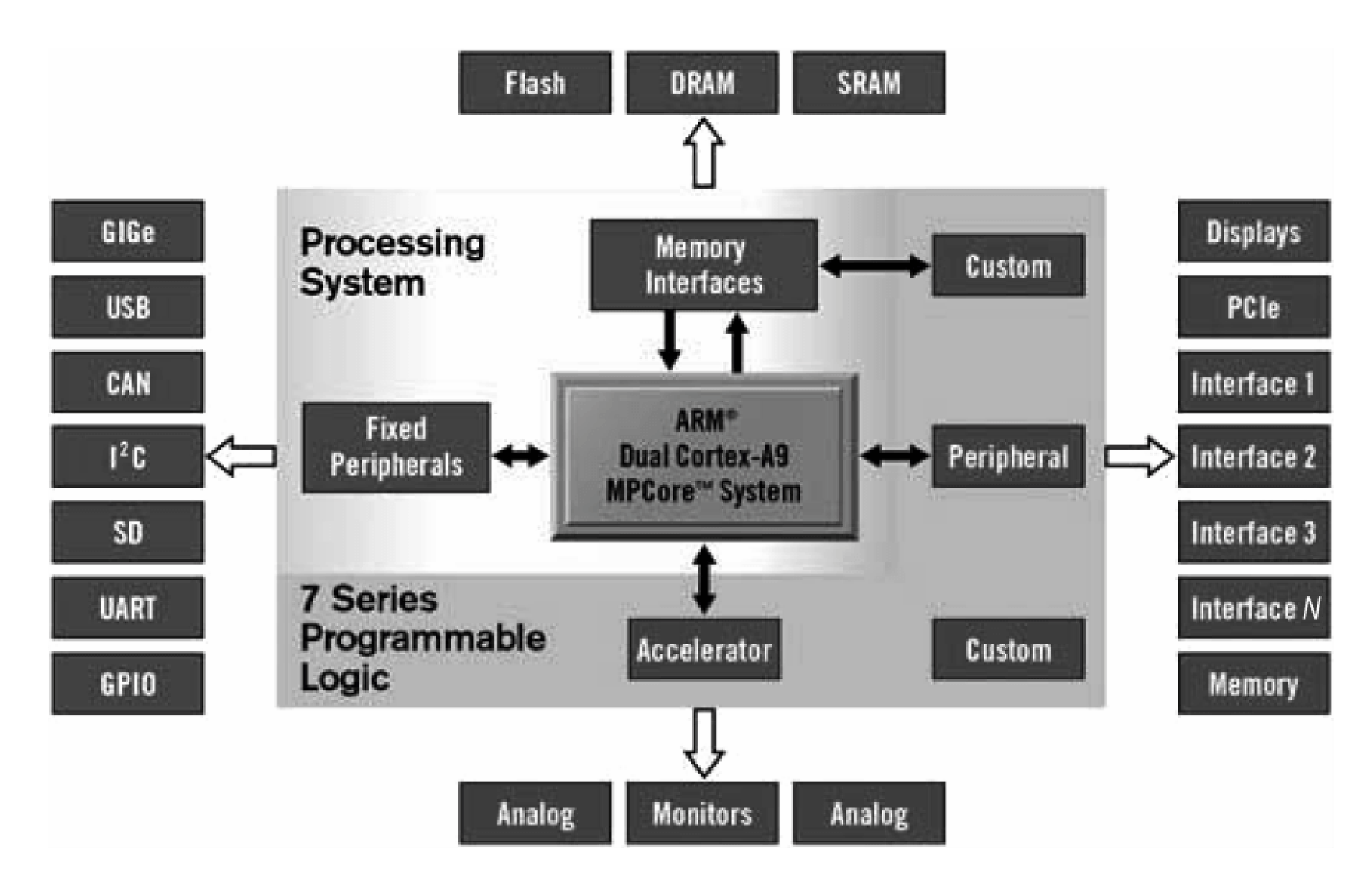


图 8-1　PYNQ 设备结构图

开源框架的硬件平台，支持 Python 编程的 PYNQ-Z1 开发板。

开发板硬件分为两部分：通用编程部分(Processing System，PS)和可编程逻辑器件部分(Programmable Logic，PL)。

1．ARM A9 CPU 部分

(1) 双核 Cortex-A9 处理器，主频为 650MHz。

(2) DDR3 内存控制器，具有 8 个 DMA 通道和 4 个高性能 AXI3 从端口。

(3) 高带宽外设控制器：1Gbps 以太网、USB 2.0、GPIO。

(4) 低带宽外设控制器：UART、CAN、I²C。

(5) 可从 JTAG、Quad-SPI 闪存和 microSD 卡进行编程。

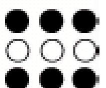
2．Artix-7 系列可编程逻辑

(1) 13 300 个逻辑片，每个具有 4 个 6 输入 LUT 和 8 个触发器。

(2) 630 KB 的快速 Block RAM；220 个 DSP 切片。

(3) 4 个时钟管理片，每个片都有一个锁相环(PLL)和混合模式时钟管理器(MMCM)。

(4) 片上模数转换器(XADC)。



8.1.3 Jupyter Notebook

PYNQ 开发板的编程环境为 Xilinx Linux 和支持 Python 编程的 Jupyter Notebook。在 ARM A9 双核 CPU 上运行的 Xilinx Linux 软件如下。

(1) Linux 内核版本 4.6.0、Python 3.6。

(2) FPGA 的基本硬件库和基于 Python 的 API 接口。

(3) 载有 Jupyter Notebook 设计环境的网络服务器。

Jupyter Notebook 是 PYNQ 板默认的开发环境，是一个基于浏览器的交互式编程计算环境，具有易用性好、用户友好的特点。在使用 Jupyter Notebook 编程时，文件里可以包含代码、画图、注释、公式、图片和视频。当 PYNQ 开发板上安装好镜像文件，就可以在 Jupyter Notebook 里轻松地用 Python 编程，使用硬件库和 Overlay 控制硬件平台。

8.2 实验设计

1. PYNQ 端

基于 PYNQ 开发，完成的功能如下。

(1) 调用 API 录音。

(2) 发送录音文件或频谱图。

2. 服务器端(Server)

基于 Flask 开发，完成的功能如下。

(1) 接收录音文件，调用服务端处理程序。

(2) 运行 Keras 或者 TensorFlow，对频谱图进行判断。

(3) 返回识别结果。

服务器的推断模型仍为 AudioNet，详见 7.1 节。

8.2.1 PYNQ 设置

PYNQ 板硬件的初始化操作过程如下。

1. 烧录 image 镜像文件

下载镜像文件并解压 PYNQ-Z1 image，版本为 v1.4 PYNQ-Z1 2017_02_10 image。

将空白的 SD 卡插入计算机，使用 win32DiskImager 烧录镜像文件。

2. 通过 USB 接口或网线连接计算机与 PYNQ 开发板

第一步，设置好跳线。

第二步，安装烧录好镜像文件的 SD 卡。

第三步，使用 Micro USB 线将 PYNQ 开发板的 PROG UART 接口连接到计算机，用来给 PYNQ 供电以及作为串口通信。

第四步，使用网线将 PYNQ 开发板连接到路由器或计算机；如果 PYNQ 通过网线连接到路由器，PYNQ 将被自动分配地址。如果 PYNQ 通过网线连接到计算机，需要先设置计算机的 IP 地址。

第五步，打开 ON 开关，等待 LED 闪动。大约一分钟后将有两个蓝色的 LED 和 4 个黄绿色的 LED 同时闪动，随后蓝色 LED 关闭，4 个黄绿色的 LED 灯亮。此时系统启动完毕，如图 8-2 所示。

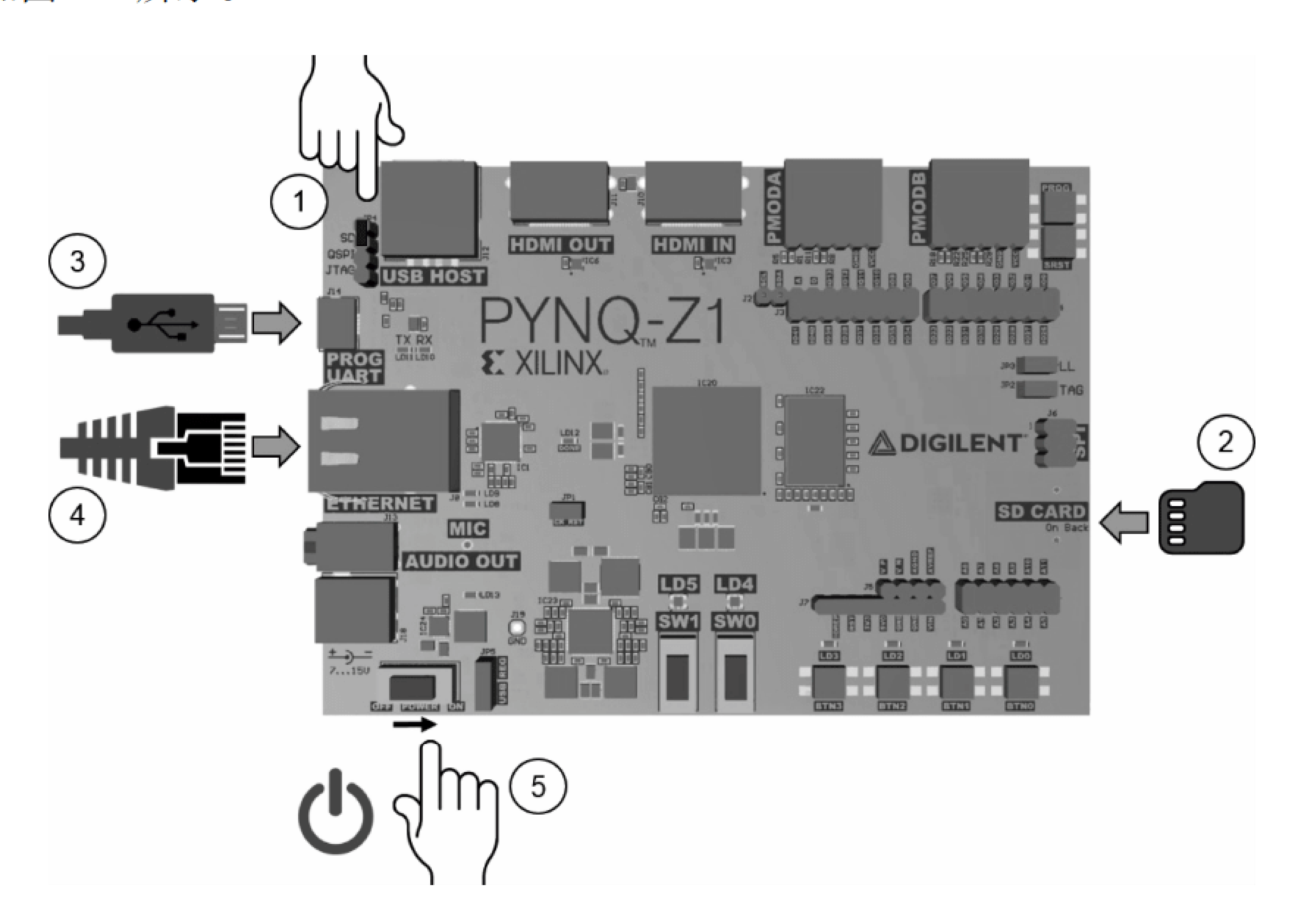
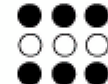


图 8-2　PYNQ 启动操作过程

3. Jupyter Notebook 浏览器编程

打开浏览器，建议采用 Chrome、Safari 或 Firefox 浏览器，IE 浏览器有兼容性问题。

如果 PYNQ 通过网线连接到路由器，PYNQ 将被自动分配地址。输入 http://



pynq:9090,输入密码 xilinx,进入 Jupyter Notebook。

如果 PYNQ 通过网线连接到计算机,需要先设置计算机的 IP 地址为 192.168.2.18,然后再打开 http://192.168.2.99:9090。同样,输入密码 xilinx,进入 Jupyter Notebook。

图 8-3 为启动 Jupyter Notebook 后在计算机上出现的运行界面。此后,就可以开始进行对 PYNQ 开发板上的资源(硬驱动和软件)进行远程操作。

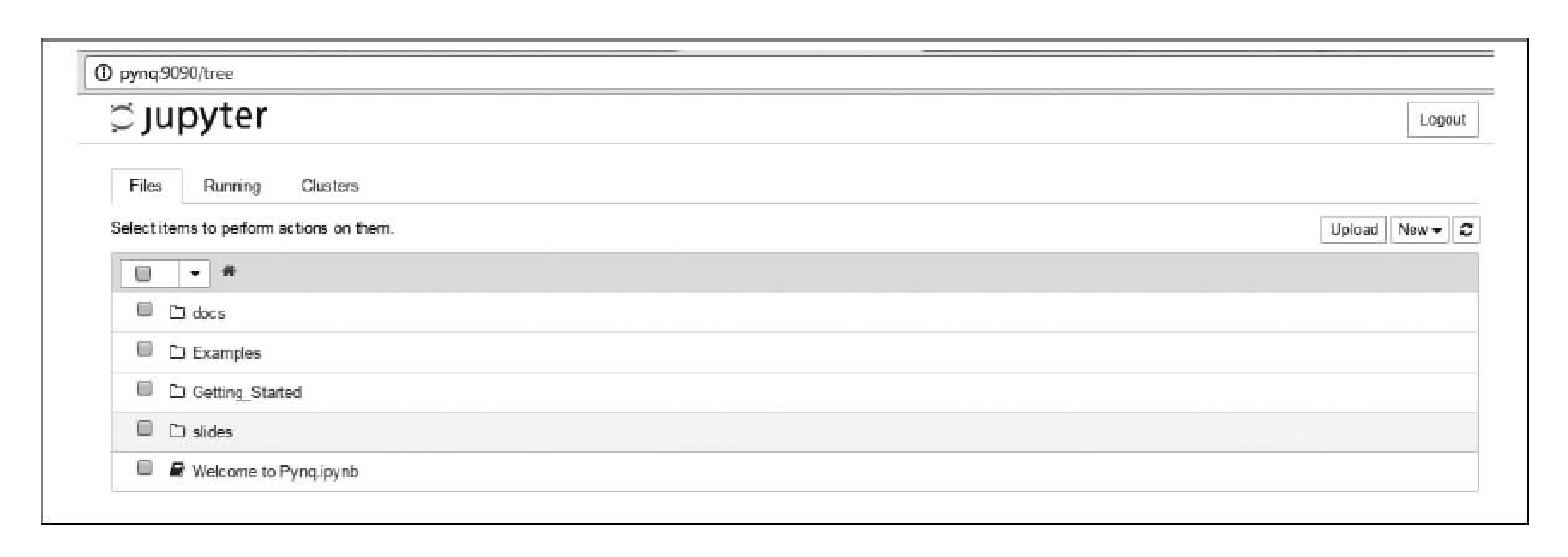


图 8-3 运行 Jupyter Notebook 后的界面

如果需要访问 PYNQ 板内文件,在 Windows 环境下,采用网络邻居访问。

```
\\pynq\xilinx
\\192.168.2.99\xilinx
```

如果是在 Linux 环境下,采用 smb 协议。

```
smb://pynq/Xilinx
smb://192.168.2.99/Xilinx
```

PYNQ 板更新可以通过执行脚本完成。该脚本会自动访问 GitHub 网站,并下载最新版本。

```
/home/xilinx/scripts/update_pynq.sh
```

8.2.2 服务器端设置

实验采用微软 Azure 云带 GPU Windows Server 虚拟主机。进入 Windows 虚拟机,通过 cmd 窗口,利用之前的数据集进行 TensorFlow 模型的训练和测试,生成一个以 .h5

为后缀的 model 文件。服务器操作如下。

（1）下载 audioNet 的文件，将训练的模型文件 *.h5 放入相应文件夹。

从 GitLab 上下载 audioNet 项目代码，网址如下：

```
http://gitlab.icenter.tsinghua.edu.cn/saturnlab/audioNet
```

（2）从网上下载 ffmpeg 的文件，放到对应的文件夹下。

augmentation	2017/5/31 16:53	文件夹	
data	2017/5/31 16:53	文件夹	
ffmpeg	2017/6/15 16:46	文件夹	
models	2017/6/15 16:44	文件夹	
sox	2017/5/31 16:53	文件夹	
tmp	2017/5/31 16:53	文件夹	
.gitignore	2017/5/31 16:53	GITIGNORE 文件	1 KB
fourierWeight.py	2017/5/31 16:53	PY 文件	2 KB
model.py	2017/5/31 16:53	PY 文件	2 KB
predict.py	2017/5/31 16:53	PY 文件	1 KB
readme.md	2017/5/31 16:53	MD 文件	1 KB
sockDataGenerator.py	2017/5/31 16:53	PY 文件	3 KB
train.py	2017/5/31 16:53	PY 文件	1 KB
vis.py	2017/5/31 16:53	PY 文件	1 KB
webfront.py	2017/6/15 16:46	PY 文件	4 KB

名称	修改日期	类型	大小
bin	2017/6/15 16:46	文件夹	
doc	2017/6/15 16:46	文件夹	
licenses	2017/6/15 16:46	文件夹	
presets	2017/6/15 16:46	文件夹	
.gitkeep	2017/5/31 16:53	GITKEEP 文件	0 KB
README.txt	2017/5/19 20:24	文本文档	4 KB

（3）打开网页 http://localhost:5000/predict，并上传音频文件进行测试。

在文件根目录下按 Shift＋右键打开 cmd 窗口，输入 activate py35 激活 Python 软件，然后调用 webfront.py 文件生成网页测试端口。操作如下：

```
#activate py35
#python webfront.py
```

该命令启动了 Flask 的 Web 服务器，简单的界面如图 8-4 所示。

在本地上传一个 *.wav 的录音文件，测试推断的正确性。

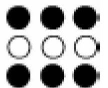




图 8-4 服务器界面

8.3 实验过程

实验过程为操作 PYNQ 板，并采用 PYNQ 板对录音进行处理，发送到具有语音识别功能的服务器上。

8.3.1 AudioInput

1. 创建新的音频项目

Create new audio object

In [8]:
```
from pynq import Overlay
from pynq.drivers import Audio

Overlay('base.bit').download()
pAudio = Audio()
```

2. 录音并将内容保存为 Recording_1. pdm

Create new audio object

In [8]:
```
from pynq import Overlay
from pynq.drivers import Audio

Overlay('base.bit').download()
pAudio = Audio()
```

Record and play

Record a 3-second sample and save it into a file.

In [8]:
```
pAudio.record(3)
pAudio.save("Recording_1.pdm")
```

Load and play

Load a sample and play the loaded sample.

In [9]:
```
pAudio.load("Recording_1.pdm")
pAudio.play()
```

3. 将录音转换成 wave 格式并发送到服务器端口进行语音识别

Step 1: Preprocessing

In this step, we first convert the 32-bit integer buffer to 16-bit. Then we divide 16-bit words (16 1-bit samples each) into 8-bit words with 1-bit sample each.

In [10]:
```
import time
import numpy as np

start = time.time()
af_uint8 = np.unpackbits(pAudio.buffer.astype(np.int16)
                         .byteswap(True).view(np.uint8))
end =  time.time()

print("Time to convert {:,d} PDM samples: {:0.2f} seconds"
      .format(np.size(pAudio.buffer)*16, end-start))
print("Size of audio data: {:,d} Bytes"
      .format(af_uint8.nbytes))
```

```
Time to convert 9,216,000 PDM samples: 0.08 seconds
Size of audio data: 9,216,000 Bytes
```

Step 2: Converting PDM to PCM

We now convert PDM to PCM by decimation. The sample rate is reduced from 3MHz to 32kHz.

We will remove the first and last 10 samples in case there are outliers introduced by decimation. We will also remove the DC offset from the waveform.

In [11]:
```
import time
from scipy import signal

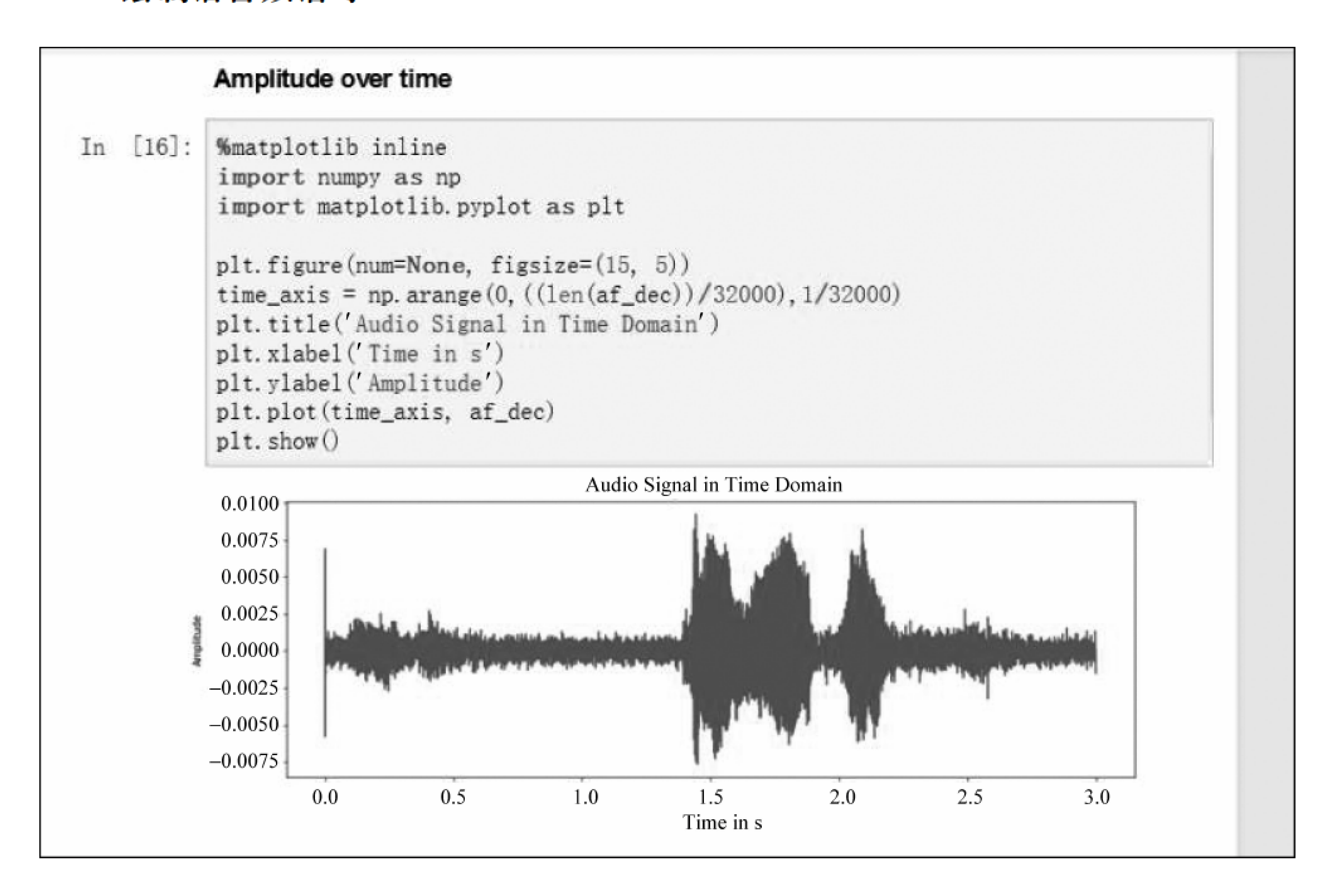
start = time.time()
af_dec = signal.decimate(af_uint8,8,zero_phase=True)
af_dec = signal.decimate(af_dec,6,zero_phase=True)
af_dec = signal.decimate(af_dec,2,zero_phase=True)
af_dec = (af_dec[10:-10]-af_dec[10:-10].mean())
end = time.time()
print("Time to convert {:,d} Bytes: {:0.2f} seconds"
      .format(af_uint8.nbytes, end-start))
print("Size of audio data: {:,d} Bytes"
      .format(af_dec.nbytes))
del af_uint8
```

```
Time to convert 9,216,000 Bytes: 6.04 seconds
Size of audio data: 767,840 Bytes
```

Step 3: Audio Playback in Web Browser

In [15]:
```
from IPython.display import Audio as IPAudio
#IPAudio(af_dec, rate=32000)
#import scipy.io.wavefile
#wavefile.write('test.wav', 32000, af_dec)
import wave
import io
#先从本地获取mp3的bytestring作为数据样本
fp=open(af_dec,rb)
data=fp.read()
fp.close()
#主要部分
aud=io.BytesIO(data)
sound=AudioSegment.from_file(aud,format='mp3')
raw_data = sound._data
#写入到文件，验证结果是否正确。
l=len(raw_data)
f=wave.open("test.wav",wb)
f.setnchannels(1)
f.setsampwidth(2)
f.setframerate(32000)
f.setnframes(1)
f.writeframes(raw_data)
f.close()
import requests
r = requests.post('101.6.161.12:5000/predict', file=open(''))
```

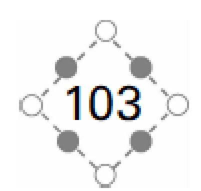
4. 绘制语音频谱等

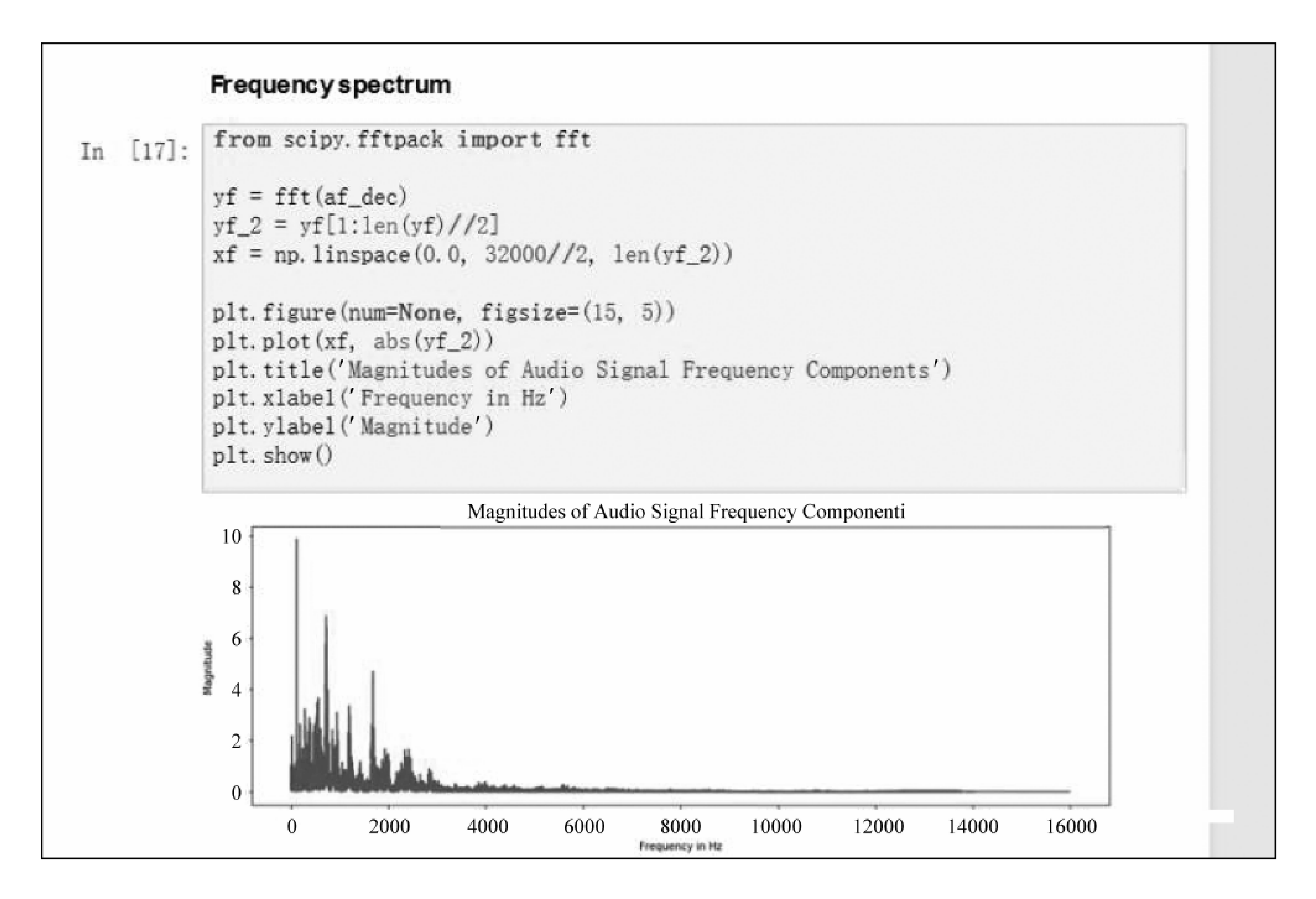


Amplitude over time

In [16]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

plt.figure(num=None, figsize=(15, 5))
time_axis = np.arange(0, ((len(af_dec))/32000), 1/32000)
plt.title('Audio Signal in Time Domain')
plt.xlabel('Time in s')
plt.ylabel('Amplitude')
plt.plot(time_axis, af_dec)
plt.show()
```



Frequency spectrum

In [17]:

```
from scipy.fftpack import fft

yf = fft(af_dec)
yf_2 = yf[1:len(yf)//2]
xf = np.linspace(0.0, 32000//2, len(yf_2))

plt.figure(num=None, figsize=(15, 5))
plt.plot(xf, abs(yf_2))
plt.title('Magnitudes of Audio Signal Frequency Components')
plt.xlabel('Frequency in Hz')
plt.ylabel('Magnitude')
plt.show()
```


Frequency spectrum over time

Use the `classic` plot style for better display.

In [18]:

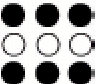
```
import matplotlib

np.seterr(divide='ignore', invalid='ignore')
matplotlib.style.use("classic")
plt.figure(num=None, figsize=(15, 4))
plt.title('Audio Signal Spectogram')
plt.xlabel('Time in s')
plt.ylabel('Frequency in Hz')
_ = plt.specgram(af_dec, Fs=32000)
```

Audio Signal Spectogram

Frequency in Hz: 0, 2000, 4000, 6000, 8000, 10000, 12000, 14000, 16000

Time in s: 0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0



8.3.2 传送云端

1. 文件格式转发

将后缀为 pdm 的文件转换为 test. wav 的波形文件，将其发送到服务器端上。

将 pdm 格式转换成更为常用的 wav 格式的代码，已经在 audioplayback. ipynb 给出示例代码，这里借助 scipy 的波形文件将已经预处理好的数据存成 wav 文件等待传输，代码如下：

```
import time
from scipy import signal
import scipy.io. wavfile as wf

start =time. time ()
af_dec =signal. decimate ( af_uint8 ,8, zero_phase =True )
af_dec =signal. decimate ( af_dec ,6, zero_phase =True )
af_dec =signal. decimate ( af_dec ,2, zero_phase =True )
af_dec =( af_dec [10: -10] -af_dec [10: -10]. mean ())
end =time . time ()

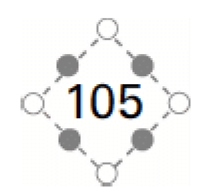
#print (" Time to convert {:,d} Bytes : {:0.2 f} seconds "
#.format ( af_uint8 . nbytes , end -start ))
#print (" Size of audio data : {:,d} Bytes "
#.format ( af_dec . nbytes ))
del af_uint8

tmp_wav_filename ='tmp ' +str( random . randint (0, 123456789)) +'. wav '
wf. write ( tmp_wav_filename , 32000 , np. array ( af_dec ))
```

2. 发送云端

采用向服务器端发送 HTTP 协议的 POST 请求的方法实现，我们使用了 Python 的 requests 库，构造了一个符合要求的 URL 格式后发出请求，因为 POST 请求会得到服务器端返回的预测结果，保存在返回值的 text 域中。代码如下：

```
import pycurl
import requests
import random
```



```
url = 'http://' + remote_ip + ':' + str(remote_port) + '/predict'
files = {'file': open(tmp_wav_filename, 'rb')}
r = requests.post(url, files=files)
predict_sentence = r.text

print(predict_sentence)
```

最后结果如下：

```
“蓝牙播放音乐”，识别正确。
```

参考文献

[1] PYNQ：Python Productivity On Zynq[EB/OL]. http://www.pynq.io.

[2] PYNQ Audio [EB/OL]. https://github.com/Xilinx/PYNQ/blob/master/Pynq-Z1/notebooks/examples/audio_playback.ipynb.

[3] PYNQ：Getting Started [EB/OL]. https://pynq.readthedocs.io.

[4] PYNQ-Z1 v1.4 Image[EB/OL]. https://github.com/Xilinx/PYNQ/tree/v1.4.



第 9 章 TX1 视觉对象检测

在第 2 章中简单介绍了各种机器视觉的对象检测(Objects Detection, OD)方法,本章将对 YOLO 系列算法的原理进行分析介绍,并在英伟达的 Jetson TX1 上进行 YOLO 算法的实践工作。

9.1 英伟达 Jetson TX1

Jetson TX1 是嵌入式模块化计算平台,可提供视觉计算应用所需的性能和节能效果。它以 Maxwell 架构为基础构建,含有 256 个 CUDA 核心,提供每秒超过 1 万亿次浮点运算的性能,功耗仅为 10～15W。

Jetson TX1 是 64 位 CPU、具有 4K 视频编解码功能,拥有 1400 兆像素/秒的相机接口,是嵌入式深度学习、计算机视觉、图形和 GPU 计算的工具。

9.2 YOLO 算法

9.2.1 YOLO 算法

YOLO 算法(You Only Look Once)是第一个达到实时性要求的深度学习检测算法,其最大的特点在于运算时的高速性(在 Titan X GPU 上的运行速度可以达到 45 帧/秒)。

YOLO 算法将过去研究者们对目标检测任务的认识由分类问题(Classification)化简

为回归问题(Regression),在保证精度不过多损失的前提下,极大地提高检测速度。

1. YOLO 的目标检测流程

YOLO 算法进行目标检测的基本流程如图 9-1 所示,详细介绍如下。

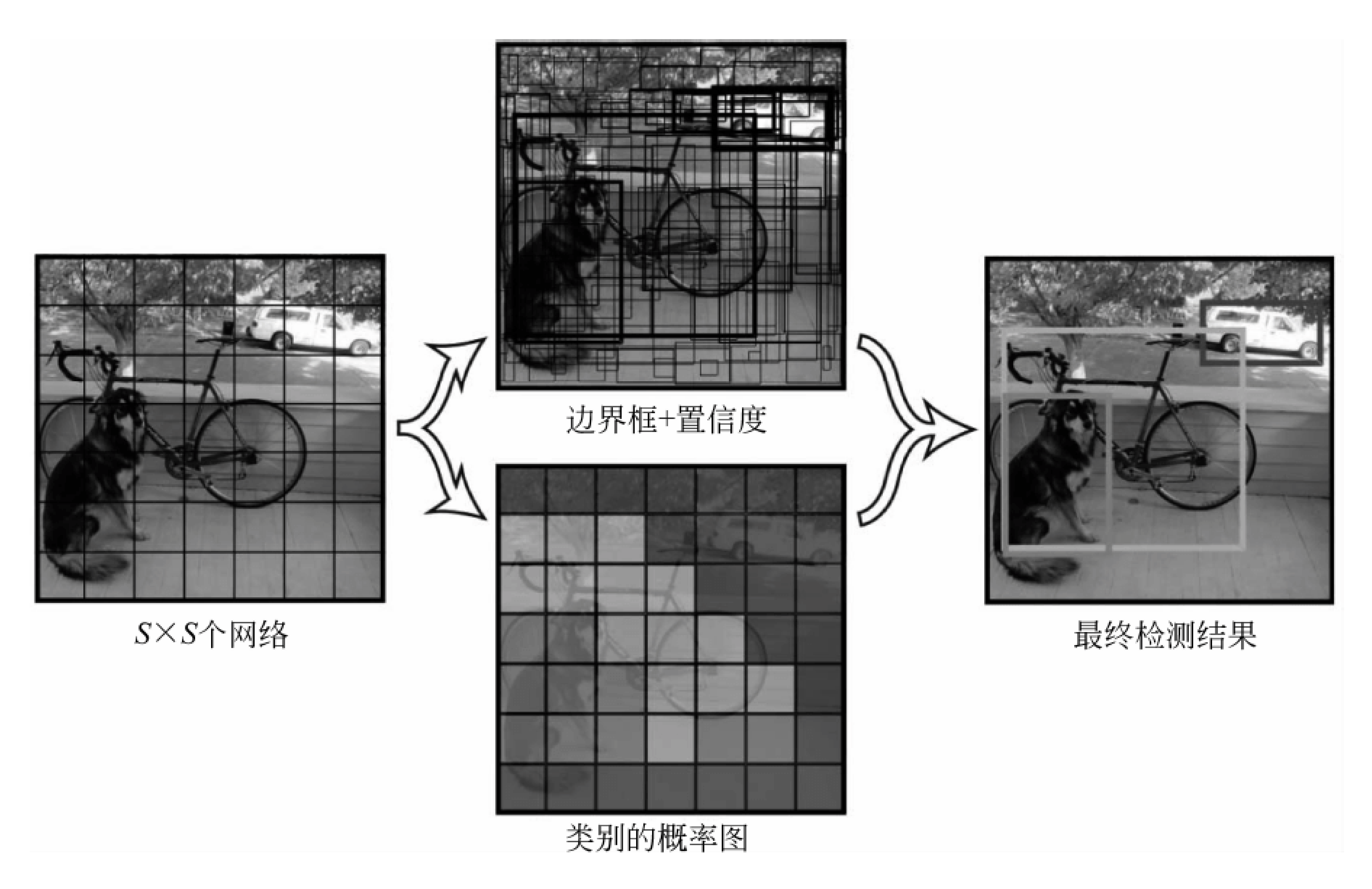


图 9-1　YOLO 算法进行目标检测的基本流程示意图

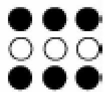
首先,将整张图像输入到神经网络中,对图像进行预处理,分割为 $S \times S$ 个网格(Grid);如果一个目标物体(Object)的中心落在某网格中,则该网格负责后续检测该目标物体的类别(Class)和位置(Location)。

随后,通过预先训练好的神经网络模型,每个网格负责预测 B 个边界框,每个边界框对应如下预测参数:边界框的中心点坐标(x、y),边界框的宽度、高度(w、h)以及置信度(Confidence)。置信度综合反映了当前边界框中存在目标的可能性 Pr(Object)以及目标位置预测的准确性$\mathrm{IOU}_{\mathrm{pred}}^{\mathrm{truth}}$,具体形式如下:

$$\text{Confidence} = \Pr(\text{Object}) \times \mathrm{IOU}_{\mathrm{pred}}^{\mathrm{truth}} \tag{9-1}$$

其中,$\mathrm{IOU}_{\mathrm{pred}}^{\mathrm{truth}}$(Intersection Over Union)表示的是预测框(Prediction)和真实框(Ground Truth)之间的重叠联合比,即两者重叠区域(交集)大小与两者联合区域(并集)大小之间的比值,数学定义如下:

$$\mathrm{IOU}_{\mathrm{pred}}^{\mathrm{truth}} = \frac{\text{Area of Intersection}}{\text{Area of Union}} \tag{9-2}$$



如果边界框中不存在目标物体，则 Pr(Object)=0，不再进行后续操作，跳至判断下一边界框。

如果存在目标，则在预测上述参数的同时，预测该网格中物体属于某一类别的条件概率 $\Pr(\text{Class}_i|\text{Object})$，取条件概率最大的类别作为我们的预测。如此就可得到每个边界框中的物体类别可能性和位置预测准确性：

$$\Pr(\text{Class}_i \mid \text{Object}) \times \Pr(\text{Object}) \times \text{IOU}_{\text{pred}}^{\text{truth}} = \Pr(\text{Class}_i) \times \text{IOU}_{\text{pred}}^{\text{truth}} \qquad (9\text{-}3)$$

最终，对置信度超过设定阈值(Threshold)的所有边界框进行非极大值抑制(Non-Maximum Suppression，NMS)去重处理，去重后得到最终的检测结果。

2. YOLO 的深度神经网络

YOLO 算法所采用的神经网络结构(见图 9-2)的设计借鉴了 GoogleNet 和 NIN 网络(Network In Network)的思想，采用 24 个卷积层后接 2 个全连接层的结构，最终输出 $S \times S \times (B \times 5 + \#\text{Classes})$ 的三维张量(Tensor)，用于进行检测判断。

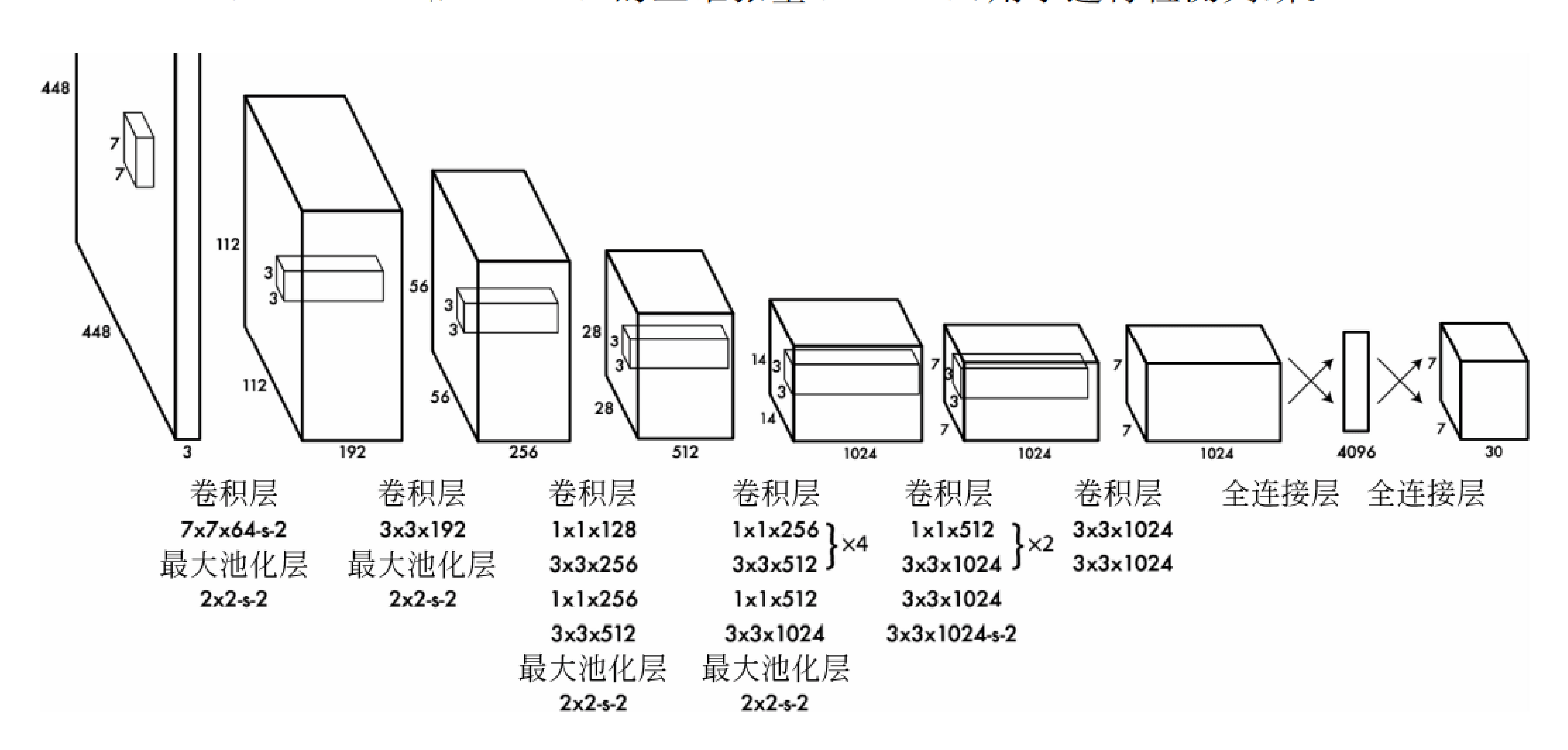


图 9-2　YOLO 算法采用的神经网络结构

YOLO 的神经网络结构如表 9-1 所示。

虽然 YOLO 算法在检测实时性方面表现出色，但是与诸如 Faster R-CNN 等高精度算法相比，其在检测结果准确性方面存在的弊端也就暴露了出来：尤其是当图像中出现成群相邻的小目标物体时，或是对于一些长宽比较为特殊的物体实例，YOLO 容易发生漏检的问题。

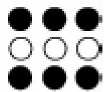
表 9-1　YOLO 的神经网络结构

No.	Type	Filters	Size / Stride	Output
1	conv	32	3×3 / 1	416×416×32
2	max		2×2 / 2	208×208×32
3	conv	64	3×3 / 1	208×208×64
4	max		2×2 / 2	104×104×64
5	conv	128	3×3 / 1	104×104×128
6	conv	64	1×1 / 1	104×104×64
7	conv	128	3×3 / 1	104×104×128
8	max		2×2 / 2	52×52×128
9	conv	256	3×3 / 1	52×52×256
10	conv	128	1×1 / 1	52×52×128
11	conv	256	3×3 / 1	52×52×256
12	max		2×2 / 2	26×26×256
13	conv	512	3×3 / 1	26×26×512
14	conv	256	1×1 / 1	26×26×256
15	conv	512	3×3 / 1	26×26×512
16	conv	256	1×1 / 1	26×26×256
17	conv	512	3×3 / 1	26×26×512
18	max		2×2 / 2	13×13×512
19	conv	1024	3×3 / 1	13×13×1024
20	conv	512	1×1 / 1	13×13×512
21	conv	1024	3×3 / 1	13×13×1024
22	conv	512	1×1 / 1	13×13×512
23	conv	1024	3×3 / 1	13×13×1024
24	conv	1024	3×3 / 1	13×13×1024
25	conv	1024	3×3 / 1	13×13×1024
26	route			
27	reorg		/ 2	13×13×2048
28	route			
29	conv	1024	3×3 / 1	13×13×1024
30	conv	55	1×1 / 1	13×13×55

9.2.2　YOLOv2 算法

1. Anchor 机制

为了克服 YOLO 算法存在的准确性问题，J. Redmon 借鉴了 Faster R-CNN 算法和



SSD 算法的思路，在原有 YOLO 算法的基础上采取了 Anchor 机制（见图 9-3(a)）来处理不同长宽比例物体的检测问题。

Anchor 机制即神经网络通过学习训练，预先设定不同边界框的长宽比的可能值，从而使得边界框能够更容易探测到长宽比相似的物体。作者在这里创新性地使用了 *K* 均值聚类（*K*-means Cluster）的方法，得到了神经网络开始训练前的最佳起始 Anchor 长宽比和数量，提升了 Anchor 机制的最终效果。

2. 新增处理技巧

YOLO 原作者 J. Redmon 还在新算法中加入了诸多其他技巧来提升算法的准确度和速度，提出了 YOLO9000。主要的改进如下。

(1) 通过直接位置预测（见图 9-3(b)），优化神经网络训练时的收敛速度；通过批次规范化（Batch Normalization）防止过拟合现象的发生。

(2) 在网络中加入转移层（Passthrough Layer），连接不同分辨率下的特征图谱（Feature Map），在增强网络检测小尺寸物体能力的同时，又不至于增加过多的计算量，保持了实时性的优势。

通过上述改进，YOLOv2 算法显著提升了检测结果的准确性，同时增快了运算速度（由原先的 45 帧/秒提升至 67 帧/秒），使得 YOLOv2 成为当前最佳的实时高精度目标检测算法，在计算速度和检测准确度之间达到一个较好的平衡。

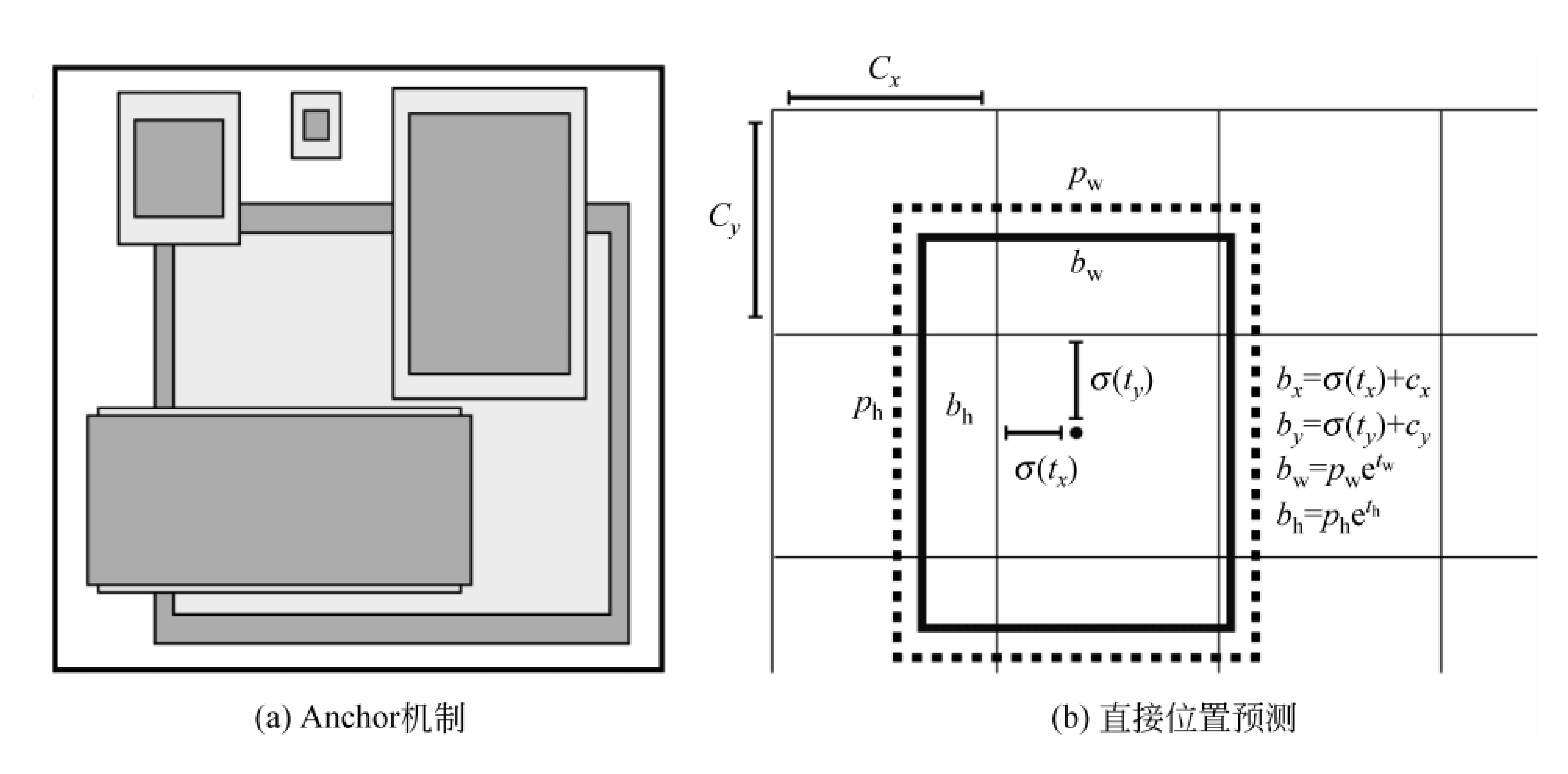


图 9-3 YOLOv2 算法新增部分技巧示例

9.2.3 YOLO 的 TX1 实践

本实验中，使用英伟达 Jetson TX1 来安装 YOLO。将 TX1 和 USB 摄像头相连，搭建 YOLO 程序环境，运行一个实时物体检测实例。

在安装之前，确保 TX1 系统中已装有 CUDA 8.0、CUDNN、OPENCV。

TX1 装机启动后，打开命令行(按 Ctrl＋Alt＋T 键)，依次输入以下命令，下载 YOLO 源代码和已经训练好的模型：

```
$git clone https://github.com/pjreddie/darknet
$cddarknet
$wget http://pjreddie.com/media/files/yolo.weights
$wget http://pjreddie.com/media/files/tiny-yolo-voc.weights
```

查看 Makefile 文件(vim Makefile)，并将前三行修改成：

```
GPU=1
CUDNN=1
OPENCV=1
```

保存后，编译 YOLO 程序：

```
$ make
```

待编译完成后，先进行单张图片的物体检测：

```
$ ./darknet detect cfg/yolo.cfgyolo.weights data/dog.jpg
```

查看并分析命令行和返回图片中的识别结果。

接着，进行实时视频的物体检测(确保已将 USB 摄像头连上主机)。由于便携式平台 TX1 的性能限制，使用常规模型的识别速度较慢，建议使用第二行命令(缩减后的模型)：

```
$ ./darknet detector demo cfg/coco.datacfg/yolo.cfgyolo.weights
```

或

```
$ ./darknet detector demo cfg/voc.datacfg/tiny-yolo-voc.cfg tiny-yolo-voc.weights
```

查看实时检测的效果(视频检测时,在命令行中输入 Ctrl+C 停止程序)。

9.3 SSD 算法

9.3.1 SSD 算法介绍

Wei Liu 等人提出了 SSD 算法。其思想是参考了 Faster R-CNN 算法中的 Anchor 机制来处理不同长宽比的物体,并通过在不同尺度的卷积特征图(Multi-scale Feature Maps)上进行检测,从而解决 YOLO 算法在小尺寸目标物体的漏检问题。

SSD 的运行速度高于 YOLO,在 TX1 板上,运行速度约 8.5 帧/秒(GPU)和约 0.03 帧/秒(CPU)。

更多的具体技术细节请参见项目网站:

```
https://myurasov.github.io/2016/11/27/ssd-tx1.html
```

9.3.2 SSD 的 TX1 实践

使用 Jetson TX1 运行 SSD 算法。预先确保系统中已装有 Ubuntu 16.04、CUDA 8.0、CUDNN、OPENCV 库。

TX1 装机启动后,打开命令行,依次输入以下命令,下载源代码和训练好的模型。

SSD 作者给出的 SSD 算法,高度依赖 Caffe 框架。使用时需要同时下载 Caffe 代码,并一同编译运行。

第一步,下载源码。

SSD 使用 Caffe 框架。

```
cd ~
mkdirssd
cdssd

#installgit
sudo apt-get update
sudo apt-get install git
```

```
#clone SSDgit repo
git clone https://github.com/weiliu89/caffe.git caffe-ssd
cdcaffe-ssd

#switch tossd branch
git checkout ssd
```

第二步，修改 Makefile. config，修改 Makefile，补充 hdf5 类库。

修改 Makefile. config 文件：

```
cd ~/ssd/caffe-ssd
cpMakefile.conig.exampleMakefile.config
```

修改 Makefile 并启用 CUDNN 类库：

```
USE_CUDNN :=1
INCLUDE_DIRS :=$(PYTHON_INCLUDE) /usr/local/include /usr/include/hdf5/serial/
```

补充 hdf5 类库。

```
LIBRARIES +=gloggflagsprotobufboost_systemboost_filesystem m hdf5_serial_hl
hdf5_serial
```

第三步，编译 Caffe。

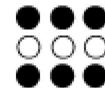
```
cd ~/ssd/caffe-ssd
make -j8
```

第四步，下载已训练好的模型，运行 SSD。

下载地址：https://myurasov. github. io/assets/posts/ssd-tx1/ssd-tx1. tar. gz。

```
#extract files
tar -zxf ssd-tx1.tar.gz
cd ssd-tx1

#run in test mode
~ubuntu/caffe-ssd/build/tools/caffe test \
  --model="test.prototxt" \
  --weights="VGG_VOC0712_SSD_300x300_iter_60000.caffemodel" \
  --iterations="536870911" \
  --gpu 0
```



第五步，下载使用已有模型。

想直接看 SSD 效果的话，可以下载训练好的模型：

下载地址：

```
https://drive.google.com/file/d/0BzKzrI_SkD1_WVVTSmQxU0dVRzA/view
```

下载后，得到了一个压缩包 models_VGGNet_VOC0712_SSD_300x300.tar.gz，将其解压，把其中的 VGGNet 文件夹放到 TX1 平台的目录之下。

```
/home/ubuntu/caffe/models/
```

第六步，配置摄像头。

修改 test.prototxt 文件：

```
/dev/video1

video_data_param {
video_type: WEBCAM
device_id: 1 <--
}
```

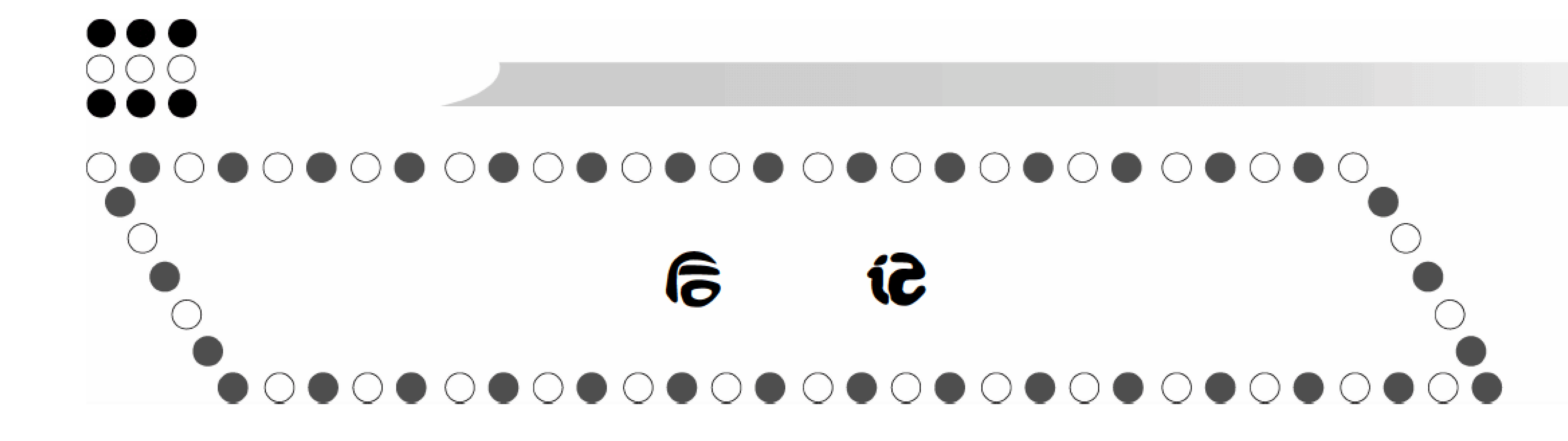
参考文献

[1] Redmon J, Divvala S, Girshick R, et al. You Only Look Once: Unified, Real-Time Object Detection[C]. Computer Vision and Pattern Recognition, 2016:779-788.

[2] Liu W, Anguelov D, Erhan D, et al. SSD: Single Shot MultiBox Detector[C]. European Conference on Computer Vision, 2016:21-37.

[3] Redmon J, Farhadi A. YOLO9000: Better, Faster, Stronger[J]. arXiv:1612.08242, 2016.

[4] Liu W, Anguelov D, Erhan D, et al. SSD: Single Shot MultiBox Detector[C]. European Conference on Computer Vision, 2016:21-37.



后　　记

阿尔法围棋的胜利，给整个社会一次人工智能技术的宣传。阿尔法围棋的成功说明用深度学习和强化学习技术，以及蒙特卡洛树搜索方法，可以完成很多规则清晰的计算机游戏的人工智能，并能有效击败人类职业选手。

阿尔法围棋的胜利，本质上也是一批计算机科学家击败了另一批职业围棋选手，是一次人类创造智能工具能力的胜利，也是科技的胜利！

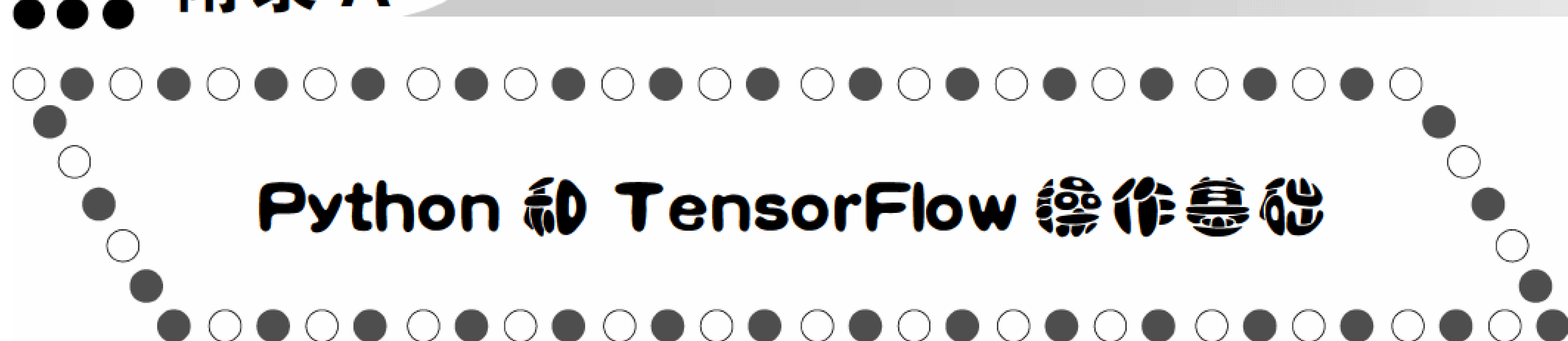
人工智能依然是一个快速进展的领域，所带来的变化也是巨大的。需要有人系统地整理、调研和实证其中的一些技术。人工智能是一个很大的领域，也是计算机的一个应用，涉及计算机体系结构、分布式系统、软硬件协同设计、算法与数据管理等，但是其核心还是算法与数据管理。本书涉及的深度学习与强化学习领域以及其实践应用也仅仅是其中很小的一部分。

人工智能需要不断创新，要求水准高，要有不断地研究探究的精神。大凡与人工智能相关的技术，都需要有训练有素的头脑。人工智能是高新技术的代表，是典型的知识经济。创业公司估值如此高，是因为其中凝聚了一只最优秀的技术团队，是技术领域的最强大脑。

人工智能领域同时也是与科研最紧密的领域之一。很多算法的研究，高校具有最优秀的头脑，因此优势巨大。同时，也要求科研工作者联系实际问题，其科研成果不仅是工业应用的产品，也是一篇篇学术研究论文。



附录 A



Python 和 TensorFlow 操作基础

Python 是一种解释型、面向对象、动态数据类型的高级程序设计语言。Python 广泛应用于数据科学与科学计算领域。

TensorFlow 是一套开源机器学习与深度网络的开发库。TensorFlow 框架提供了最全面的 Pythor 语言 API 接口。

本书安装的软件是 Python 3.6.2 和 TensorFlow 1.3。默认操作系统为 Windows 10。

A.1 Python 实践基础

1. Python 系统目录和文件名操作

```
import os
os.listdir()
os.path.join()
os.path.basename()
```

(1) 字符串类型的 split()方法，通过指定分隔符对字符串进行分隔。

```
str=('24_0.png')
str1=str.split('.')[0]
str2=str1.split('_')
print(str2[1])
```

(2) 字符串 str 转换成 int：int_value =int(str_value)。

```
a=int(str2[1])
```

（3）glob 模块用于逐个获取匹配文件的路径名。

```
import glob
glob.glob(r'./*.py')
```

```
f=glob.iglob(r"./*.*")
for py in f:
  print(py)
```

2．Python 音频波形和频谱图

（1）打开 Juypter notebook 或者 IPython。

```
$juypter notebook
```

或者

```
$ipython notebook
```

查看 IPython 的工作路径：

```
import os
os.path.realpath('.')
```

（2）用录音程序生成 a.m4a 文件，进行音频文件的转化。

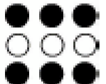
使用 ffmpeg 工具，将不同格式的音频文件归一化。

```
os.system('.\\ffmpeg\\bin\\ffmpeg.exe -i .\\a.m4a -ac 1 -acodec pcm_f32le -ar
44100 .\\a.wav -v 1')
```

（3）生成语音波形图。

```
import matplotlib.pyplot as plt
from wavReader import readWav
rate, data =readWav('./a.wav')
plt.plot(data)
plt.show()
```

（4）生成频谱图。



```
import os
from matplotlib import image, pyplot
os.path.exists('.\\sox\\sox.exe')
os.system('.\\sox\\sox.exe .\\a.wav -n rate 4k spectrogram -o aout.png')
img =image.imread('.\\aout.png')
pyplot.imshow(img)
pyplot.axis('off')
pyplot.show()
```

3. Python 实践：sigmoid、softmax 和 argmax 函数

用 Python 绘制机器学习有几个重要函数，即 sigmoid、softmax 和 argmax 函数等。

绘图时需要 matplotlib(见 http://matplotlib.org/)。

运算时需要 numpy(见 https://docs.scipy.org/doc/numpy-dev/user/quickstart.html)。

1) sigmoid 函数

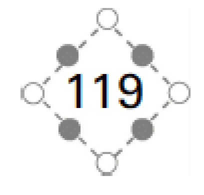
```
import matplotlib.pyplot as plt
import numpy as np
x =np.linspace(-5,5,5)
sigmoid =lambda x: 1 / (1 +np.exp(-x))
plt.plot(x,sigmoid(x), color='red', lw=2)
plt.show()
```

2) softmax 函数

```
import math
w=[1,2,3,4,5,6,7,8,9]
w_exp=[math.exp(i) for i in w]
print(w_exp)
sum_w_exp =sum(w_exp)
softmax =[round(i / sum_w_exp, 3) for i in w_exp]
print(softmax)
print(sum(softmax))
```

3) argmax 函数

输入一个 one_shot 向量(只有一个元素为 1，其余元素为 0，见 https://docs.scipy.org/doc/numpy/reference/generated/numpy.argmax.html)。



```
import numpy as np
z=[1,2,3,6,5]
print(np.argmax(z))
w=np.arange(8).reshape(2,4)
np.argmax(w)
```

A.2 TensorFlow 实践基础

1. TensorFlow 的变量与矩阵定义

```
import tensorflow as tf
import numpy
A=tf.Variable([[1,2],[3,4]], dtype=tf.float32)
A.get_shape()
```

```
TensorShape([Dimension(2), Dimension(2)])
```

```
B=tf.Variable([[5,6],[7,8]],dtype=tf.float32)
B.get_shape()
```

```
TensorShape([Dimension(2), Dimension(2)])
```

```
C=tf.matmul(A,B)
tf.global_variables()
```

```
[<tf.Variable 'Variable:0' shape=(2, 2) dtype=float32_ref>, <tf.Variable
'Variable_1:0' shape=(2, 2) dtype=float32_ref>]
```

2. TensorFlow 变量实例化和矩阵相乘

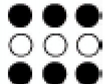
TensorFlow 变量在初始化后，会赋予一个指定的操作。只有在 sess.run()时，才真正有数值，并进行运算。

```
init =tf.global_variables_initializer()
sess =tf.Session()
sess.run(init)
print(sess.run(C))
```

```
[[ 19.  22.]
[ 43.  50.]]
```

```
print(sess.run(tf.reduce_sum(C, 0)))
```

```
[ 62.  72.]
```



```
print(sess.run(tf.reduce_sum(C, 1)))
[ 41.  93.]
```

3. TensorFLow 的 sigmoid、softmax 和 softplus 函数

这些函数已经被有效地封装到 TensorFlow 库中(见 https://www.tensorflow.org/api_guides/python/nn)。

```
R=tf.Variable([1.,.2,3],dtype=tf.float32)
T=tf.Variable([True, True, False, False], dtype=tf.bool)
U=tf.Variable([True, True, True, False], dtype=tf.bool)
init=tf.global_variables_initializer()
sess.run(init) /sess.run(tf.global_variables_initializer())
tf.variables_initializer(T).run()
print(sess.run(tf.sigmoid(R, name="last")))
print(sess.run(tf.softmax(R,dim=-1, name="last")))
print(sess.run(tf.nn.softplus(R,name="last")))
print(sess.run(tf.nn.relu(R,name="XXX")))
print(sess.run(tf.cast(T, tf.int32)))
print(sess.run(tf.cast(T, tf.float32)))
print(sess.run(tf.reduce_mean(tf.cast(T, tf.float32))))
```

4. TensorFLow 网站的教程链接

1) TensorFlow 线性模型

见 https://www.tensorflow.org/tutorials/wide。

2) TensorFlow 逻辑斯提回归

见 https://www.tensorflow.org/tutorials/wide#defining_the_logistic_regression_model。

3) TensorFlow DNN 分类器(Iris 数据集)

见 https://www.tensorflow.org/get_started/tflearn。

4) TensorFlow CNN 分类器(MINIST 数据集)

见 https://www.tensorflow.org/tutorials/layers。

5) TensorFlow 深度卷积网络(CIFAR 数据集)

见 https://www.tensorflow.org/tutorials/deep_cnn。